Министерство образования и науки Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

С.П. Куксенко

Электромагнитная совместимость: вычислительные методы

Учебно-методическое пособие

Томск – 2017

Куксенко С.П.

Электромагнитная совместимость: вычислительные методы. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2017. – 163 с.

Представлены методические материалы практикума, посвященного изучению вычислительных методов, инструментальных средств, и средств валидации, применяемых при моделировании задач электромагнитной совместимости. Работы выполняются в программных продуктах TALGAT и GNU Octave (или Scilab).

Предназначено для студентов, магистрантов и аспирантов технических вузов, специализирующихся в области электромагнитной совместимости.

Содержание

Введение

В общем случае, в основе моделирования лежит численный анализ, требующий построения математической модели исследуемого объекта с помощью решения уравнений Максвелла. Процесс построения математической модели можно разбить на следующие этапы: постановка задачи – определение целей расчета, объема необходимой входной и выходной информации и допустимой погрешности. Аналитическая обработка – формулировка начальных и граничных условий, описание параметров расчетной области, выбор метода решения. Дискретизация (сегментация) модели – переход от непрерывных функций к дискретным и от функциональных уравнений к системе линейных алгебраических уравнений (СЛАУ). Вычисление элементов СЛАУ. Решение СЛАУ – выбор наиболее подходящего метода решения (прямой или итерационный). Обработка результатов (вычисление требуемых характеристик) – вычисление интересующих характеристик из решения СЛАУ.

Важно, что перечисленные этапы не являются независимыми. Так, выбор метода дискретизации (сегментации) влияет на затраты на формирование СЛАУ и на размер и свойства получаемой СЛАУ, что, в свою очередь, определяет выбор метода её решения. От предыдущих этапов зависят и способы вычисления параметров и характеристик исследуемого объекта/системы.

Таким образом, от умения выбирать наиболее предпочтительный численный метод и его программное реализовывать зависит эффективность всего процесса моделирования.

1 Практические занятия

1.1 Программные среды GNU Octave и Scilab

1.1.1 GNU Octave

1.1.1.1 Основы синтаксиса, базовые операции и функции

GNU Octave (далее по тексту Octave) – это свободно распространяемый язык программирования высокого уровня, ориентированный на проведение численных расчетов, и, по сути, являющийся альтернативой коммерческому пакету MATLAB. Octave обладает богатым инструментарием для решения задач линейной алгебры, нахождения корней нелинейных уравнений, решения дифференциальных уравнений, вычисления интегралов, решения линейных и нелинейных оптимизационных задач, построения графиков и т.д. Только некоторые из этих возможностей Octave будут описаны в настоящем пособии.

После запуска Octave (версия 4.2.1) пользователь видит окно интерпретатора (рисунок 1.1). Перечислим основные инструменты интерфейса пользователя:

1. Область главного меню. Позволяет получить доступ ко всем возможным командам и интерфейсам.

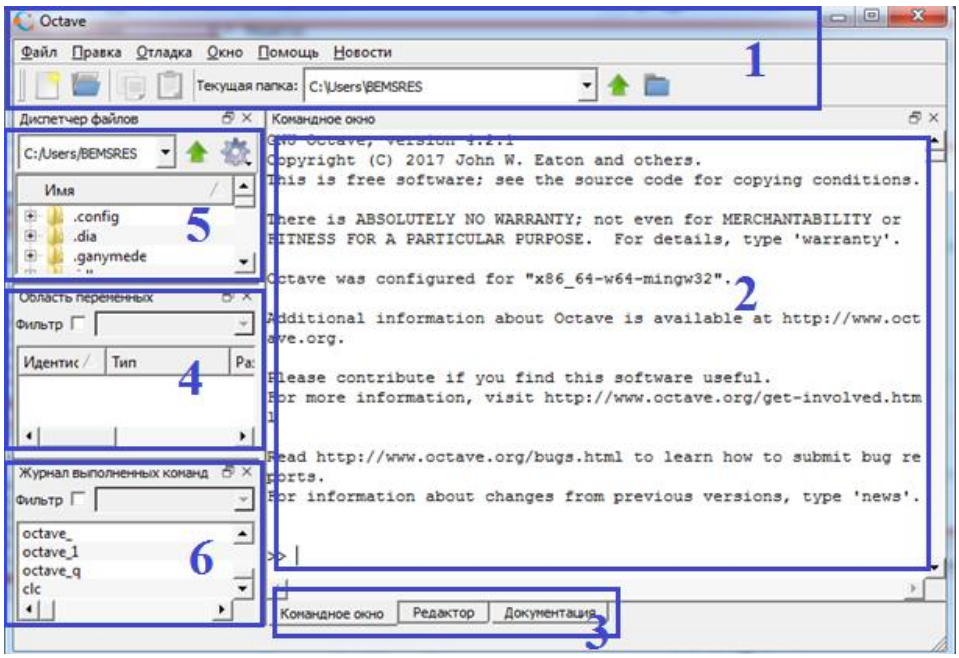


Рисунок 1.1 – Графическое окно Octave

2. Область редактора. Позволяет вносить необходимые изменения в программы, созданные пользователем или импортированные им из внешних источников. Здесь под программой понимается текстовый файл, содержащий команды для среды Octave с расширением .m. Содержащиеся в файлах команды последовательно передаются на исполнение системой через командное окно.

3. Командное окно/Редактор. В окне командное окно (интерпретатор) пользователь может вводить как отдельные команды языка Octave, так и группы команд. Группы команд удобно объединять в программы (окно Редактор). Содержащиеся в файлах команды последовательно передаются на исполнение системой через Командное окно (результат выполнения программы также выводится в командном окне).

4. Область переменных. В этой области пользователь может следить за значением и типом данных переменных. Такой контроль позволяет контролировать правильность расчетов.

5. Файловый менеджер.

6. Журнал выполненных команд. Содержит список команд и информацию об их выполнении.

При необходимости пользователь может закрывать ненужные ему окна либо открывать новые (см. вкладка *Окно* области главного меню).

Возможны два варианта решения любой задачи в Octave:

1. Терминальный режим (вкладка *Командное окно*). В этом режиме в окно интерпретатора последовательно вводятся отдельные команды.

2. Программный режим (вкладка *Редактор*). В этом режиме создаётся текстовый файл с расширением .m, в котором хранятся последовательно выполняемые команды Octave. Затем эта программа запускается на выполнение.

Для примера создадим две матрицы и перемножим их, сначала в терминальном режиме. (Вычислить значение выражения можно, если ввести его в командную строку и нажать клавишу ENTER). Пример реализации

приведен на рисунке 1.2. В переменной *ans* хранится результат последней операции, если команда не содержит знака присваивания (её значение изменяется после каждого вызова команды без операции присваивания).

```
>> a=[2 3; 4 5]%create matrix
a =

   2   3
   4   5

>> [4 5; 9 8]%create matrix
ans =

   4   5
   9   8

>> ans*a
ans =

   28   37
   50   67
```

Рисунок 1.2 – Перемножение двух матриц: терминальный режим

Если строка заканчивается символом «;», результаты на экран не выводятся. Если же в конце строки этот символ отсутствует, то результаты работы выводятся на экран. Текст в строке после символа «%» является комментарием и интерпретатором не обрабатывается.

Теперь рассмотрим, как решить эту же задачу в программном режиме. Для этого нужно перейти во вкладку *Редактор* и последовательно ввести команды:

```
a=[2 3; 4 5];
b=[4 5; 9 8]
b*a
```

После чего сохранить файл, например, с именем example1.m в нужной директории рабочей станции и нажать F5. Далее перейдя назад во вкладку *Командное окно* можно увидеть результат вычисления, рисунок 1.3.

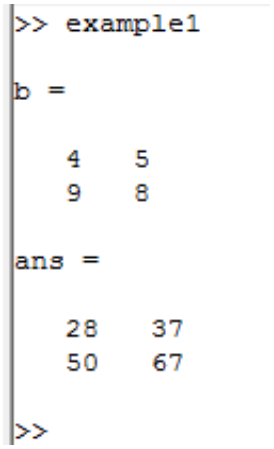


Рисунок 1.3 – Перемножение двух матриц: программный режим

Далее рассмотрим основные моменты синтаксиса языка Octave (демонстрация примеров выполнена в терминальном режиме). Основной рабочей конструкцией (структурой) в Octave, является матрица. Это не означает, что нельзя выполнять операции с вещественными или комплексными числами, просто нужно хорошо понимать, что таким числам соответствуют матрицы размерности 1×1. Рассмотрим основные математические операции.

Операция присваивания «=» передает значение выражения в левой части равенства переменной, стоящей справа. Так, напишем в командной строке (далее для облегчения восприятия все приводимые примеры будут показаны с помощью курсивного выделения)

a=5;

Результат выполнения команды не будет отражен в командном окне, так как в конце выражения стоит оператор отключения вывода «;». При вводе данного выражения в командную строку переменной *a* будет присвоено значение 5. Тем не менее, *a* с точки зрения интерпретатора является матрицей размером 1×1. Убедиться в этом можно, дав команду (или воспользоваться областью переменных)

>> size(a)
ans =
1 1

Т.е. команда *size*(*a*) предназначена для определения размерность матрицы *a*.

Имя переменной не должно совпадать с именами встроенных процедур,

функций и встроенных переменных системы. Система различает большие и малые буквы в именах переменных, т.е. ABC, abc, Abc, aBc и тд. – это имена разных переменных.

Если команда не содержит знака присваивания, то по умолчанию вычисленное значение присваивается специальной системной переменной *ans*. Причем полученное значение можно использовать в последующих вычислениях, но важно помнить, что значение *ans* изменяется после каждого вызова команды без оператора присваивания.

Кроме переменной *ans* в Octave существуют и другие системные переменные, которые можно использовать в математических выражениях:

- i, j – мнимая единица;
- pi – число π;
- *e* –экспонента, 2.71828183;
- realmin – наименьшее число с плавающей точкой (2.2251e-308);
- realmax – наибольшее число с плавающей точкой (1.7977e+308).
- inf – машинный символ бесконечности;
- NaN – неопределенный результат (0/0 или ∞/∞ и тд.).

Как было показано выше, возможно выполнение операции присвоения сразу ко всем элементам матрицы:

```
>> B=[1,2,3]
B =
   1 2 3
```

Команда формирует вектор-строку (матрицу размерности 1×3). При введении данных через «;» будет сформирован вектор-столбец:

```
>> C=[4;5;6]
C =
   4
   5
   6
```

Аналогичным образом, формируются матрицы любой размерности.

Если значения некоторой величины меняются с равным шагом, то оператор присваивания может быть использован совместно с оператором «:»,

определяющим пределы изменения величины. Так, шаблон команды будет выглядеть следующим образом: ПЕРЕМЕННАЯ=НАЧАЛЬНОЕ ЗНАЧЕНИЕ:ШАГ:КОНЕЧНОЕ ЗНАЧЕНИЕ. Значение шага также задается пользователем, но при его отсутствии он полагается равным 1. Например,

```
>> E=1:4:17
E =
    1 5 9 13 17.
```

Оператор «:» имеет еще одно применение – он может определять совокупность строк или столбцов в матрице. Для демонстрации сначала создадим нулевую матрицу *D* размера 4×4:

```
>> D=zeros(4,4)
D =
    0  0  0  0
    0  0  0  0
    0  0  0  0
    0  0  0  0
```

Затем выполним команду

```
>> D(1,:)=1
D =
    1  1  1  1
    0  0  0  0
    0  0  0  0
    0  0  0  0
```

Так, с помощью оператора «:» произошло присвоение значения 1 всем элементам первой строки матрицы *D*. Еще один пример использования данного оператора приведен ниже.

```
>> D(3:4,2:3)=-1
D =
    1  1  1  1
    0  0  0  0
    0 -1 -1  0
    0 -1 -1  0
```

Для очистки командного окна можно воспользоваться командой *clc*, а для удаления из памяти всех переменных или конкретных переменных командами *clear* и *clear имя_переменной* соответственно. Простейшие арифметические операции в Octave выполняют с помощью следующих операторов:

+ сложение;

– вычитание;

* умножение;

/ деление слева направо;

\ деление справа налево;

^ возведение в степень.

Если вычисляемое выражение слишком длинное, то перед нажатием клавиши ENTER следует набрать три или более точек. Это будет означать продолжение текущей (командной) строки:

```
>> 1+2+...
4+5
ans =  12
```

Допускается переназначение переменных не только на уровне значений, но и на уровне типов данных. Например, вещественной переменной можно присвоить символьное значение – при такой операции автоматически произойдет смена ее типа. Интерпретаторы команд отслеживают только согласование размерности в выражении (умножение символьной строки на матрицу будет вызывать ошибку). Пример: действие оператора присваивания:

```
>> A=[3 4; 7 5];
>> A='line';
>> B=[1 2; 3 4];
>> A*B
error: operator *: nonconformant arguments (op1 is 1x4, op2 is 2x2)
```

При использовании матричного типа данных нужно помнить, что при этом невозможно, например, выполнить сложение или вычитание для матриц разной размерности. Требования к размерности матриц также ограничивают применение операций умножения. Для удобства пользователей некоторые математические операции могут быть также выполнены поэлементно. Указание на поэлементное выполнение дается добавлением знака «.» к оператору. Так, рассмотрим пример.

```
>> A=[1,2;3,4];
>> B=A*A
```

```
B =
      7 10
      15 22
>> C=A.*A
C =
      1 4
      9 16
```

В первом случае матрица **A** была умножена сама на себя (выполнено матричное умножение), во втором случае сам на себя был умножен каждый элемент матрицы (поэлементное возведение в квадрат).

Числовые результаты могут быть представлены с плавающей (3.2E-9, 6.42E+3), или с фиксированной (4.12, 17.1389) точкой. Числа в формате с плавающей точкой представлены в экспоненциальной форме $mE \pm p$, где m – мантисса (целое или дробное число с десятичной точкой), p – порядок (целое число). Для того чтобы перевести число в экспоненциальной форме к обычному представлению с фиксированной точкой, необходимо мантиссу умножить на десять в степени порядок. Далее приведен пример использование различных комбинаций задания вещественного числа.

```
>> 5.49e-2
ans =  0.054900
>> 5.49E-2
ans =  0.054900
>> 5.49.*1E-2
ans =  0.054900
>> 5.49.*10^-2
ans =  0.054900
>> 5.49*10^-2
ans =  0.054900
```

Рассмотрим пример ввода числа.

```
>> 0.987654321
ans =  0.98765
```

Видно, что число знаков в дробной части числа в строке ввода больше чем в строке вывода. Происходит это потому, что результат вычислений выводится в виде, который определяется предварительно установленным форматом представления чисел. Команда, с помощью которой можно установить формат числа имеет вид: *format* формат числа. В Octave предусмотрены следующие

форматы чисел (приведены используемые в ходе занятий):

- Short – краткая запись, применяется по умолчанию.
- Long – длинная запись.
- Short E (Short e) – краткая запись в формате с плавающей точкой.
- Long E (Long e) – длинная запись в формате с плавающей точкой.

Продемонстрируем использование форматов на примере вывода числа π.

```
>> format short
>> pi
ans =  3.1416
>> format long
>> pi
ans =  3.14159265358979
>> format short E
>> pi
ans =   3.1416E+000
>> format long E
>> pi
ans =   3.14159265358979E+000
```

Имеются также тригонометрические функции и экспоненциальные функции exp(x) – экспонента числа x и log(x) – натуральный алгоритм числа *x*. Рассмотрим пример.

```
>> x=pi/2
>> x =  1.5708
>> sin(x)
ans =  1
>> cos(x)
ans =   6.1230e-017
>> tan(x)
ans =   1.6331e+016
>> exp(x)
ans =  4.8105
>> log(x)
ans =  0.45158
```

Также имеются целочисленные функции:

- fix(x) – округление числа *x* до ближайшего целого в сторону нуля;
- floor(x) – округление числа *x* до ближайшего целого в сторону отрицательной бесконечности;

- ceil(x) – округление числа *x* до ближайшего целого в сторону положительной бесконечности;
- round(x) – округление числа *x* до ближайшего целого;
- rem(x, y) – вычисление остатка от деления *x* на *y*;
- sign(x) – выдает 0, если *x*=0, –1 при *x* < 0 и 1 при *x* > 0;

и другие элементарные функции:

- sqrt(x) – корень квадратный из числа *x*;
- abs(x) – модуль числа *x*;
- log10(x) – десятичный логарифм от числа *x*;
- log2(x) – логарифм по основанию два от числа *x*;
- pow2(x) – возведение двойки в степень *x*;
- gcd(x, y) – наибольший общий делитель чисел *x* и *y*;
- lcm(x, y) – наименьшее общее кратное чисел *x* и *y*;
- rats(x) – представление числа *x* в виде рациональной дроби.

Рассмотрим реализацию комплексных чисел. Как было отмечено выше для обозначения мнимой единицы используются *i* и/или *j*. Ввод комплексного числа производится в следующем формате:

действительная_часть + i*мнимая_часть.

К комплексным числа применимы элементарные арифметические операции: +, –, *, \, /, ^. Функции для работы с комплексными числами:

- real(z) – выдает действительную часть комплексного аргумента *z*;
- imag(z) – выдает мнимую часть комплексного аргумента *z*;
- angle(z) – вычисляет значение аргумента комплексного числа *z* в радианах от $-\pi$ до π;
- conj(z) – выдает число комплексно сопряженное Z.

Рассмотрим операции отношения, предназначены для выполнения сравнения двух операндов и определяют, истинно выражение или ложно. Результат операции отношения – логическое значение. В качестве логических значений используются 1 («истина») и 0 («ложь»). Операции отношения:

- < – меньше;
- > – больше;
- == – равно;
- ~= – не равно;
- <= – меньше или равно;
- >= – больше или равно.

В Octave существует возможность представления логических выражений в виде логических операторов и логических операций(таблица 1.1).

Таблица 1.1 – Виды логических выражений

Тип выражения	Выражение	Логический оператор	Логическая операция
Логическое «и»	A and B	and(A, B)	A & B
Логическое «или»	A or B	or(A, B)	A | B
Исключающее «или»	A xor B	xor(A,B)	
Отрицание	not A	not (A)	~ A

Для визуализации входных данных и полученных результатов Octave располагает обширными библиотеками графических построений. Возможно построение как двумерных, так и трехмерных графиков. Основной функцией для построения двумерных графиков служит функция plot, у которой несколько вариантов вызова:

- plot(X,Y) – построение графика зависимости $y(x)$. Значения y и x берутся из матриц **Y** и **X**, которые могут быть либо вектором-столбцом, либо вектором-строкой одинаковой размерности;
- plot(X1,Y1,..., Xn,Yn) – одновременное построение нескольких функциональных зависимостей $y(x)$, при этом параметры линий на графике установлены по умолчанию;
- plot(X,Y,LineSpec), plot(X1,Y1,LineSpec1,..., Xn,Yn,LineSpecn) – наиболее полный вариант вызова функции построения двумерных графиков с заданием параметров графических линий. LineSpec – это шаблон, с помощью которого определяется цвет линии, ее толщина, вид маркеров и

другие параметры. Шаблон представляет собой взятое в апострофы название параметра, отделенное запятой от его значения. Подробное описание параметров и значений можно посмотреть в документации, ниже приведена таблица 1.2 с основными параметрами.

Таблица 1.2 – Виды логических выражений

Параметр	Возможные значения
Color' – цвет линии	'y' – желтый 'm' – магента (пурпурный) 'c' – циан (зелено-голубой) 'r' – красный 'g' – зеленый 'b' – голубой 'k' – черный [r g b] – цвет в формате *RGB* (параметры *r*, *g*, *b* лежат в пределах от 0 до 1)
LineStyle' – вид линии	'-' – сплошная '- -' – двойная сплошная ':' – пунктирная '-.' – штрихпунктирная 'none' – без линии (будут оставлены только маркеры)
'LineWidth' – ширина линии	Положительные значения с шагом 0.5
Marker' – вид маркера	'o' – круговой '+' – знак «+» 's' – квадрат 'p' – пятиугольник '^' – ориентированный вверх треугольник

Пример построения графика sin(*x*):

```
>> x=0:0.1:2*pi;
>> plot(x,sin(x),'Color','m','LineStyle','-.','LineWidth',4);
```

Если применить еще одну команду plot, то после ее выполнения график новой функции будет нарисован поверх предыдущего и затрет его. Данная особенность связана не с работой функции plot, а с работой графического окна. Поэтому все функции, связанные с построением графиков, применяются обычно в паре с функциями, определяющими режим работы графического

окна. Некоторые из них приведены в таблице 1.3.

Таблица 1.3 – Основные параметры функции plot

Функция	Варианты использования
figure – функция запроса графического окна. После применения все изображения будут перенаправляться в запрошенное окно	figure() – вызывает графическое окно, номер которому будет присвоен автоматически (возможен вызов в виде N=figure(), в этом случае переменной *N* будет присвоен номер графического окна) figure(N) – будет создано новое окно с номером присвоен номер *N*, если такого окна не существовало, в противном случаев в него будет перенаправлен графический вывод
hold – функция, включающая и отключающая сохранение в графическом окне предыдущих графиков	hold on – новые графики будут изображаться совместно с предыдущими hold off – новые графики будут изображаться поверх предыдущих, затирая их
grid – функция, включающая и отключающая отображение линий сетки	grid on – включает отображение линий сетки grid off – выключает отображение линий сетки
xlabel, ylabel – функции подписывания осей	xlabel('НАЗВАНИЕ ОСИ X') ylabel('НАЗВАНИЕ ОСИ Y')
legend – функция нанесения на график легенды (сопровождающих и поясняющих надписей)	legend(' СТРОКА 1', ' СТРОКА 2', ...)

Далее приведен пример построение зависимостей sin(*x*) и cos(*x*) на одном графике.

```
>> x=0:0.1:2*pi;
>> figure(1);
>> hold on;
>> plot(x,sin(x),'Color','m','LineStyle','-.','LineWidth',4)
>> plot(x,cos(x),'Color','g','LineStyle','--','LineWidth',3)
>> legend('sin(x)','cos(x)')
>> grid on;
>> xlabel('x');
>> ylabel('sin(x)');
```

Octave предоставляет возможность построить несколько осей в графическом окне и вывести на каждую из них свои графики. Для этого следует использовать функцию:

>> subplot(row, col, cur);

Параметры row и col определяют количество графиков по вертикали и горизонтали соответственно, cur определяет номер текущего графика. Повторное обращение к функции subplot с теми же значениями row и col позволяет просто изменять номер текущего графика и может использоваться для переключения между графиками.

Для построения графиков поверхностей используется несколько функций. График поверхности (трехмерный или 3D-график) – это график, положение точки в котором определяется значениями трех координат. Одной из функций является mesh(). Перед её описанием рассмотрим функцию meshgrid(). Данная функция формирует прямоугольную сетку. Так, выполнив команду

>> [x y]=meshgrid(-2:2,-1:1)

Получим

```
x =
    -2  -1   0   1   2
    -2  -1   0   1   2
    -2  -1   0   1   2
y =
    -1  -1  -1  -1  -1
     0   0   0   0   0
     1   1   1   1   1
```

Далее воспользуемся собственно функцией mesh(), но предварительно вычислим значения функции во всех узловых точках. Для конкретики рассмотрим следующий пример.

```
>> z=4*x.^2-2*sin(y).^2
z =
    14.58385   2.58385  -1.41615   2.58385  14.58385
    16.00000   4.00000   0.00000   4.00000  16.00000
    14.58385   2.58385  -1.41615   2.58385  14.58385
```

Далее идет вызов функции mesh(x, y, z) для построения «каркасного» графика.

Еще один из способов построения трехмерных графиков связан с использованием функции surf(), строящей «каркасную» поверхность. Наиболее часто функция вызывается в формате surf(**X**, **Y**, **Z**) или в surf(**X**, **Y**, **Z**, *C*), где **X** и **Y** – векторы-строки, определяющие значения абсцисс и ординат, а **Z** – матрица с размерностью, равной произведению размерностей матриц **X** и **Y**,

задающая значения координаты *z*, для соответствующих пар *x* и *y*. Параметр *C* определяет способ отображения трехмерной картинки (цвет, режим отображения кромок и т. д.). Данная функция будет использована в курсе при вычислении дивергенции, ротора и др.

Перечисленные выше способы построения как двумерных, так и трехмерных графиков не являются единственно возможными. О других функциях и способах можно узнать из документации в ходе самостоятельных работы.

1.1.1.2 Работа с матрицами, матричные операции

Элементы вектора-строки отделяют пробелами или запятыми, а всю конструкцию заключают в квадратные скобки:

```
>> a=[2 -3 5 6 -1 0 7 -9]
a =
     2 -3 5 6 -1 0 7 -9
>> b=[-1,0,1]
b =
     -1 0 1
```

Вектор-столбец можно задать, если элементы отделять друг от друга точкой с запятой:

```
>> c=[-pi;-pi/2;0;pi/2;pi]
c =
     -3.14159
     -1.57080
     0.00000
     1.57080
     3.14159
```

Обратится к элементу вектора можно, указав имя вектора, а в круглых скобках номер элемента под которым он хранится в этом векторе:

```
>> a(1)
ans = 2
>> b(3)
ans = 1
>> c(5)
ans = 3.1416
```

Ввод элементов матрицы так же осуществляется в квадратных скобках,

при этом элементы строки отделяются друг от друга пробелом или запятой, а строки разделяются между собой точкой с запятой:

```
>> Matr=[0 1 2 3;4 5 6 7]
Matr =
     0 1 2 3
     4 5 6 7
```

Обратиться к элементу матрицы можно, указав после имени матрицы, в круглых скобках, через запятую, номер строки и номер столбца, на пересечении которых элемент расположен:

```
>> Matr(2,3)
ans = 6
>> Matr(1,1)
ans = 0
>> Matr(1,1)=pi; Matr(2,4)=-pi;
>> Matr
Matr =
     3.1416 1.0000 2.0000 3.0000
     4.0000 5.0000 6.0000 -3.1416
```

Матрицы и векторы можно формировать, составляя их из ранее заданных матриц и векторов:

```
>> a=[-3 0 2];b=[3 2 -1];c=[5 -2 0];
>> %Горизонтальная конкатенация векторов–строк, результат вектор–
строка
>> M=[a b c]
M = -3 0 2 3 2 -1 5 -2 0
>> %Вертикальная конкатенация векторов–строк, результат матрица
>> N=[a;b;c]
N =
     -3 0 2
     3 2 -1
     5 -2 0
>> %Горизонтальная конкатенация матриц
>> Matrica=[N N N]
Matrica =
     -3 0 2 -3 0 2 -3 0 2
     3 2 -1 3 2 -1 3 2 -1
     5 -2 0 5 -2 0 5 -2 0
>> %Вертикальная конкатенация матриц

>> Tablica=[M;M;M]

Tablica =
```

```
    -3 0 2 3 2 -1 5 -2 0
    -3 0 2 3 2 -1 5 -2 0
    -3 0 2 3 2 -1 5 -2 0
```

Важную роль при работе с матрицами играет знак двоеточия «:»:

```
>>Tabl=[-1.2 3.4 0.8;0.9 -0.1 1.1;7.6 -4.5 5.6;9.0 1.3 -8.5]
Tabl =
    -1.20000 3.40000 0.80000
    0.90000 -0.10000 1.10000
    7.60000 -4.50000 5.60000
    9.00000 1.30000 -8.50000
>> %Выделить из матрицы 3-й столбец
>> Tabl(:,3)
ans =
    0.80000
    1.10000
    5.60000
    -8.50000
>> %Выделить из матрицы 1-ю строку
>> Tabl(1,:)
ans = -1.20000 3.40000 0.80000
>> %Выделить из матрицы подматрицу
>> Matr=Tabl(2:3,1:2)
Matr =
     0.90000 -0.10000
     7.60000 -4.50000
>> %Вставить подматрицу в нижний угол исходной матрицы
>> Tabl(3:4,2:3)=Matr
Tabl =
    -1.20000 3.40000 0.80000
    0.90000 -0.10000 1.10000
    7.60000 0.90000 -0.10000
    9.00000 7.60000 -4.50000
>> %Удалить из матрицы 2-й столбец
>> Tabl(:,2)=[]
Tabl =
    -1.20000 0.80000
    0.90000 1.10000
    7.60000 -0.10000
    9.00000 -4.50000
>> %Удалить из матрицы 2-ю строку
>> Tabl(2,:)=[]
Tabl =
    -1.20000 0.80000
    7.60000 -0.10000
```

```
    9.00000 -4.50000
>>%Представить матрицу в виде вектора–столбца
>> Matr
Matr =
    0.90000 -0.10000
    7.60000 -4.50000
>> Vector=Matr(:)
Vector =
    0.90000
    7.60000
    -0.10000
    -4.50000
>> %Выделить из вектора элементы со 1-го по 3-й
>> V=Vector(1:3)
V =
    0.90000
    7.60000
    -0.10000
>> %Удалить из массива 2-й элемент
>> V(2)=[]
V =
    0.90000
    -0.10000
```

Рассмотрим действия над векторами предусмотренные в Octave. Для сложения векторов используют знак «+». Операция сложения определена только для векторов одного типа, то есть суммировать можно либо векторы-столбцы, либо векторы-строки одинаковой длины. (Вычитание векторов выполняется с помощью знака «–».)

```
>> a=[2 4 6];b=[1 3 5];
>> c=a+b
c =
    3 7 11
```

Знак апострофа «'» применяется для транспонирования вектора:

```
>> a'
ans =
    2
    4
    6
```

Умножение вектора на число осуществляется с помощью знака «*». Знак деления «/» применяют для того, чтобы разделить вектор на число:

```
>>> z=2*a+a/4
z =
    4.5000 9.0000 13.5000
```

Умножение вектора на вектор выполняется так же с помощью знака «*». Перемножать можно только векторы одинакового размера, причем один из них должен быть вектором-столбцом, а второй вектором-строкой.

```
>> a=[2 4 6]; b=[1 3 5];
>> %В результате умножения вектора-строки  на вектор–столбец получится число
>> a*b'
ans = 44
>> %В результате умножения вектора-столбца на вектор–строку получится матрица
>> a'*b
ans =
    2 6 10
    4 12 20
    6 18 30
>> % Некорректное умножение векторов
>> a*b
error: operator *: nonconformant arguments (op1 is 1x3, op2 is 1x3)
```

Все перечисленные действия над векторами определены в математике и относятся к так называемым векторным вычислениям. Но Octave допускает и поэлементное преобразование векторов. Существуют операции, которые работают с вектором не как с математическим объектом, а как с обычным одномерным массивом. Например, если к некоторому заданному вектору применить математическую функцию, то результатом будет новый вектор того же размера и структуры, но элементы его будут преобразованы в соответствии с заданной функцией:

```
>> x=[-pi/2,-pi/3,-pi/4,0,pi/4,pi/3,pi/2]
x =
    -1.5708 -1.0472 -0.7854 0.0000 0.7854 1.0472 1.5708
>> y=sin(2*x)+cos(2*x)
y =
    -1.0000 -1.36603 -1.0000 1.0000 1.0000 0.36603 -1.0000
>> y=2*exp(x/5)
y = 1.4608 1.6221 1.7093 2.0000 2.3402 2.4660 2.7382
```

Поэлементное умножение векторов выполняется при помощи оператора

«.*». В результате формируется вектор, каждый элемент которого равен произведению соответствующих элементов заданных векторов:

```
>> a=[2 4 6];b=[1 3 5];
>> a.*b
ans = 2 12 30
```

Поэлементное деление одного вектора на другой осуществляется при помощи следующей конструкции «./». В результате получается вектор, каждый элемент которого частное от деления соответствующего элемента первого вектора на соответствующий элемент второго. Совокупность знаков «.\» применяют для деления векторов в обратном направлении (поэлементное деление второго вектора на первый). Пример деления векторов:

```
>> a=[2 4 6];b=[1 3 5];
>> a./b
ans =
    2.0000 1.3333 1.2000
>> a.\b
ans =
    0.50000 0.75000 0.83333
```

Поэлементное возведение в степень выполняет оператор «.^»,

Далее рассмотрим действия над матрицами. Одним из базовых действий над матрицами является сложение «+» (вычитание «–»). Важно помнить, что суммируемые (вычитаемые) матрицы должны быть одной размерности.

Умножать на число «*» можно любую матрицу, результатом так же будет матрица, каждый элемент которой будет помножен на заданное число. Операция транспонирования «'» меняет в заданной матрице строки на столбцы и также применима к матрицам любой размерности. При умножении матриц «*» важно помнить, что число столбцов первой перемножаемой матрицы должно быть равно числу строк второй.

Возведение матрицы в степень «^» эквивалентно ее умножению на себя указанное число раз. При этом целочисленный показатель степени может быть как положительным, так и отрицательным. Матрица в степени –1 называется обратная к данной. При возведении матрицы в положительную степень выполняется алгоритм умножения матрицы на себя указанное число раз.

Возведение в отрицательную степень означает, что умножается на себя матрица обратная к данной.

Для поэлементного преобразования матриц (листинг 5.24) можно применять операции, описанные ранее, как операции поэлементного преобразования векторов: добавление (вычитание) числа к каждому элементу матрицы «+» («–»), поэлементное умножение матриц «.*» одинакового размера, поэлементное деление матриц одинакового размера (прямое «./» и обратное «.\»), поэлементное возведение в степень «.^» и применение к каждому элементу матрицы математических функций. Рассмотрим пример использования описанных операций.

```
>> A=[1 2 3; 4 5 6; 7 8 9]; B=[-1 -2 -3; -4 -5 -6; -7 -8 -9];
>> (2*A+1/4*B')^2-A*B^(-1)
ans =
    83.875   93.125  104.375
   219.250  244.062  272.875
   356.625  395.000  457.375
```

Оператор «/» используется для операции называемой делением матриц слева направо, соответственно «\» применяется для деления матриц справа налево. Операция *B/A* эквивалентна выражению $\mathbf{B}\cdot\mathbf{A}^{-1}$, ее удобно использовать для решения матричных уравнений вида $\mathbf{X}\cdot\mathbf{A} = \mathbf{B}$. Соответственно *A\B* эквивалентно $\mathbf{A}^{-1}\cdot\mathbf{B}$ и применяется для решения уравнения $\mathbf{A}\cdot\mathbf{X} = \mathbf{B}$.

Обозначим через **x** и **b** векторы, а матрицу через **A**, то получим запись системы линейных алгебраических уравнений в матричной форме $\mathbf{Ax} = \mathbf{b}$. Это значит, что оператор «\» можно применять для решения линейных систем.

```
>> A=[1 2;1 1];
>> b=[7;6];
>> x=A\b
x =
   5
   1
```

В Octave существуют специальные функции, предназначенные для работы с матрицами и векторами. Эти функции можно разделить на следующие группы: функции для работы с векторами; функции для работы с матрицами;

функции, реализующие численные алгоритмы решения задач линейной алгебры. Далее рассмотрим наиболее часто используемые функции для работы с векторами:

- length(X) – определяет длину вектора **X**;
- prod(X) – вычисляет произведение элементов вектора **X**;
- sum(X) – вычисляет сумму элементов вектора **X**;
- diff(X) – формирует вектор, размер которого на единицу меньше чем у вектора **X**, а каждый элемент представляет собой разность между двумя соседними элементами массива;
- min(X) – находит минимальный элемент вектора **X**, вызов в формате [nomX, nom]=min(X) дает возможность определить минимальный элемент *nomX* и его номер *nom* в массиве **X**;
- max(X) – находит максимальный элемент массива **X** или при [nomX,nom]=max(X) определяет максимум и его номер;
- mean(X) – определяет среднее арифметическое массива **X**;
- dot(x1,x2) – вычисляет скалярное произведение двух векторов;
- cross(x1,x2) – определяет векторное произведение векторов;
- sort(X) – выполняет сортировку массива **X** (сортировка по возрастанию sort(X), сортировка по убыванию -sort(-X)).

Далее рассмотрим наиболее часто используемые функции для работы с матрицами:

- eye(n [, m]) – возвращает единичную матрицу (вектор) соответствующей размерности);
- ones((n [, m, p, …]) – формирует матрицу (вектор), состоящую из единиц;
- zeros(n [, m, p, …]) – возвращает нулевую матрицу (вектор) соответствующей размерности;
- diag(X [, k]) – возвращает квадратную матрицу с элементами X на главной диагонали или на *k*-й; функция diag(M [, k]), где **M** ранее

определенная матрица, в качестве результата выдаст вектор столбец, содержащий элементы главной или *k*-й диагонали матрицы **M**;

- rand([n, m, p, …]) – возвращает матрицу (вектор), элементы которой – случайные числа, распределенные по равномерному закону, rand без аргументов возвращает одно случайно число;
- randn([n, m, p…]) – возвращает матрицу (вектор), элементы которой – случайные числа распределенные по нормальному закону; randn без аргументов – возвращает одно случайно число;
- linspace(a, b [, n]) – возвращает массив из 100 или *n* точек, равномерно распределенных между значениями *a* и *b*;
- logspace(a, b [, n]) – формирует массив из 50 или *n* точек, равномерно распределенных в логарифмическом масштабе между значениями 10*a* и 10*b*; (функция logspace(a, pi) дает равномерное распределение из 50 точек в интервале от 10*a* до π);
- repmat(M, n [, m]) – формирует матрицу, состоящую из *n*×*n* или *n*×*m* копий матрицы **M** (если *M* – скаляр, то формируется матрица, элементы которой равны значению *M*);
- reshape(M, m, n) – возвращает матрицу размерности *m*×*n*, сформированную из **M** путем последовательной выборки по столбцам (если **M** не содержит *m* на *n* элементов, то выдается сообщение об ошибке);
- cat(n, A, B, [C, …]) – объединяет матрицы **A** и **B** или все входящие матрицы, если *n* = 1, слияние матриц происходит по столбцам, если *n* = 2 – по строкам;
- rot90(M [, k]) – осуществляет поворот матрицы **M** на 90 градусов или на величину 90*k*, где *k* – целое число;
- tril(M [, k]) – формирует из матрицы **M** нижнюю треугольную матрицу начиная с главной или с *k*-й диагонали;

- triu(M [, k]) – формирует из матрицы **M** верхнюю треугольную матрицу начиная с главной или с *k*-й диагонали;
- size(M) – определяет число строк и столбцов матрицы **M**;
- prod(M [,k]) – формирует вектор-строку или вектор-столбец, в зависимости от значения *k*, каждый элемент которого является произведением элементов соответствующего столбца (*k* = 1) или строки (*k* = 2) матрицы **M** (если значение параметра *k* в конструкции отсутствует, то по умолчанию вычисляются произведения столбцов матрицы);
- sum(M [,k]) – формирует вектор-строку или вектор-столбец, в зависимости от значения *k*, каждый элемент которого является суммой элементов соответствующего столбца (*k* = 1) или строки (*k* = 2) матрицы **M** (если значение параметра *k* в конструкции отсутствует, то по умолчанию вычисляются суммы столбцов матрицы);
- diff(M) – из матрицы **M** размерностью $n \times m$ формирует матрицу, размером $n-1 \times m$ элементы которой представляют собой разность между элементами соседних строк матрицы **M**;
- min(M) – формирует вектор-строку, каждый элемент которого является наименьшим элементом соответствующего столбца матрицы **M**, определить положение этих элементов в матрице можно, если вызвать функцию в формате [n, m]=min(M), где *n* – это вектор минимальных элементов столбцов матрицы **M**, а *m* – вектор номеров строк матрицы **M**, в которых находятся эти элементы, конструкция min(min(M)) позволит отыскать минимум среди всех элементов матрицы; вызов функции в виде min(M, [], k) или [n, m]=min(M, [], k) позволяет управлять направлением поиска, в частности можно отыскать минимальные элементы и их положение в строках матрицы **M**; и наконец функция min(A, B) сформирует матрицу из строк min(A) и min(B);
- max(M) – формирует вектор-строку, каждый элемент которого является наибольшим элементом соответствующего столбца матрицы **M**, действия

функций [n, m]=max(M), max(max(M)), max(M, [], k), [n, m]= max(A, [], k), max(A,B) аналогичны действиям семейства функций min();

- mean(M, [k]) – формирует вектор-строку или вектор-столбец, в зависимости от значения *k*, каждый элемент которого является средним значением элементов соответствующего столбца или строки матрицы **M**, если значение параметра *k* в конструкции отсутствует, то по умолчанию вычисляются средние значения столбцов матрицы; среднее всех элементов матрицы вычисляет функция mean(mean(M));
- sort(M) – выдает матрицу того же размера, что и M, каждый столбец которой упорядочен по возрастанию;
- sqrtm(A) – возвращает матрицу **X**, для которой **X**·**X** = **A**;
- expm(A) и logm(A) – взаимообратные матричные функции, первая вычисляет матричную экспоненту $e^{\mathbf{A}}$, а вторая выполняет логарифмирование по основанию *e*.

Далее рассмотрим наиболее часто используемые функции реализующие численные алгоритмы решения задач линейной алгебры:

- det(M) – вычисляет определитель квадратной матрицы **M**;
- trace(M) – вычисляет след матрицы **M**, то есть сумму элементов главной диагонали;
- norm(M [, p]) – возвращает различные виды норм матрицы **M** в зависимости от *p*, если аргумент *p* = 1, 2, inf, fro (если p не задан, то вычисляется вторая норма матрицы **M**);
- cond(M [, p]) – возвращает число обусловленности матрицы **M**, основанное на норме *p*;
- rcond(M) – вычисляет величину обратную значению числа обусловленности матрицы относительно первой нормы, если полученная величина близка к единице, то матрица хорошо обусловлена, если к приближается к нулю, то плохо;
- inv(M) – возвращает матрицу обратную к **M**;

- eig(M) – возвращает вектор собственных значений матрицы **M**, вызов функции в формате [Matr, D]=eig(M) даёт матрицу **Matr**, столбцы которой собственные векторы матрицы **M** и диагональную матрицу **D** , содержащую собственные значения матрицы **M**; функция eig(A, B), где **A** и **B** квадратные матрицы, выдает вектор обобщенных собственных значений;
- rref(M) – осуществляет приведение матрицы **M** к треугольной форме, используя метод исключения Гаусса;
- chol(M) – возвращает разложение по Холецкому для положительно определенной симметричной матрицы **M**; функция L=chol(M) возвращает верхнюю треугольную матрицу **L**, для которой $\mathbf{M} = \mathbf{L}^T \cdot \mathbf{L}$, функция L=chol(M,'lower') возвращает нижнюю треугольную матрицу **L**, для которой $\mathbf{M} = \mathbf{L} \cdot \mathbf{L}^T$; матрица **L** в обоих случаях со строго положительными элементами на главной диагонали;
- lu(M) – выполняет LU-разложение, функция [L, U, P]=lu(M) возвращает три матрицы: **L** – нижняя треугольная, **U** – верхняя треугольная и **P** – матрица перестановок, причем $\mathbf{P} \cdot \mathbf{M} = \mathbf{L} \cdot \mathbf{U}$; функция lu(M) без параметров возвращает одну матрицу, которая в свою очередь, является комбинацией матриц **L** и **U**;
- qr((M) – выполняет QR–разложение, команда [Q, R, P]=qr(M) возвращает три матрицы: ортогональную матрицу **Q**, верхнюю треугольную матрицу **R** и матрицу перестановок **P**, причем $\mathbf{M} \cdot \mathbf{P} = \mathbf{Q} \cdot \mathbf{P}$;
- svd(M) – возвращает вектор сингулярных чисел матрицы, при использовании в формате
- [U, S, V]=svd(M) выполняет сингулярное разложение матрицы **M**, выдает три матрицы: **U** – сформирована из ортонормированных собственных векторов, отвечающих наибольшим собственным значениям матрицы $\mathbf{M} \cdot \mathbf{M}^T$, **V** – состоит из ортонормированных собственных векторов матрицы $\mathbf{M} \cdot \mathbf{M}^T$, **S** – диагональная матрица из сингулярных чисел

(неотрицательных значений квадратных корней из собственных значений матрицы $\mathbf{M}\cdot\mathbf{M}^T$), матрицы удовлетворяют условию $\mathbf{M} = \mathbf{U}\cdot\mathbf{S}\cdot\mathbf{V}^T$.

1.1.1.3 Разреженные матрицы

Под разреженными матрицами принято понимать совокупность схемы хранения данных в сочетании с соответствующим алгоритмом для выполнения требуемой операции. Если предложенные схема хранения данных и алгоритм позволяют получить выигрыш по требуемой памяти хранения и времени выполнения по сравнению со стандартной схемой хранения в виде массива и обычным алгоритмом, тогда стоит говорить о разреженных матрицах. Такие матрицы возникают в различных прикладных задачах. В частности при решении электромагнитных задач, решаемых такими численными методами как метод конечных разностей и моментов, речь о которых пойдет в последующих разделах.

Разреженные матрицы хранятся вOctave в компактной форме, подразумевающей хранение только ненулевых элементов и соответствующих индексов строк и столбцов. Проще всего понять схему хранения, обратившись к функции sparse, одно из назначений которой – преобразование обычного формата в разреженный формат хранения:

```
>> A=[1 0 0; 0 2 1; 3 0 4]
A =
   1   0   0
   0   2   1
   3   0   4

>> sparse(A)
ans =
Compressed Column Sparse (rows = 3, cols = 3, nnz = 5 [56%])
(1, 1) ->  1
(3, 1) ->  3
(2, 2) ->  2
(2, 3) ->  1
(3, 3) ->  4
```

Преимущества такой схемы хранения очевидны: нужна память только для ненулевых элементов типа double и индексов типа uint32. Соответственно,

массив **A** занимает 72 байта, а для хранения **S** требуется 64 байта. Для создания разреженной матрицы вовсе не требуется, как в предыдущем примере, создавать сначала двумерный массив, а затем преобразовывать его функцией sparse. Можно сразу указать ненулевые элементы и их расположение в матрице (и размер, поскольку последние строки или столбы могут быть полностью нулевые):

```
>> S=sparse([1,3,2,2,3],[1,1,2,3,3],[1,3,2,1,4],3,3)
S =
Compressed Column Sparse (rows = 3, cols = 3, nnz = 5 [56%])
  (1, 1) ->  1
  (3, 1) ->  3
  (2, 2) ->  2
  (2, 3) ->  1
  (3, 3) ->  4
```

Для обратного преобразования (и проверки) служит функция full

```
>> A=full(S)
A =
   1   0   0
   0   2   1
   3   0   4
```

Если ненулевые элементы матрицы сосредоточены на определенных диагоналях, то для создания таких матриц предназначена функция spdiags. Для пользователя реализация матричных операций скрыта, а все операции осуществляются при помощи обычных символов: *, +, -, ^ и поэлементных операций. При этом в результате получаются разреженные матрицы. Более того, в матричное выражение могут одновременно входить как разреженные, так и обычные матрицы, соответствующие следующим правилам:

- разреженная + разреженная $\Rightarrow$ разреженная;
- разреженная – разреженная $\Rightarrow$ разреженная;
- разреженная * разреженная $\Rightarrow$ разреженная;
- разреженная .* разреженная $\Rightarrow$ разреженная;
- разреженная ./ разреженная $\Rightarrow$ разреженная;
- разреженная + плотная $\Rightarrow$ плотная;

- разреженная – плотная ⇒ плотная;
- разреженная * плотная ⇒ плотная;
- разреженная .* плотная ⇒ разреженная;
- разреженная ./ плотная ⇒ разреженная.

Для визуализации разреженных матриц применяется функция spy, которая выводит портрет разреженной матрицы, рисунок 1.4.

```
>> S=sprand(100,100,0.01);
>> spy(S)
```

Для определения количества ненулевых элементов матрицы служит команда nnz()

```
 >> nnz(S)
ans =  200
```

1.1.1.4 Основы программирования

Рассмотренные в предыдущих подразделах группы команд, состоящие из операторов присваивания и обращения к встроенным функциям, представляют собой простейшие программы Octave. Если такая программа хранится в файле с расширением .m (.M), то для ее выполнения достаточно в командной строке Octave ввести имя этого файла (без расширения). Рассмотрим основные операторы этого языка и примеры их использования.

Даже при разработке простейших программ возникает необходимость ввода исходных данных и вывода результатов. Если для вывода результатов на экран можно просто не ставить «;» после оператора, то для ввода исходных данных при разработке программ, работающих в диалоговом режиме, следует использовать функцию

имя_переменной = input('подсказка');

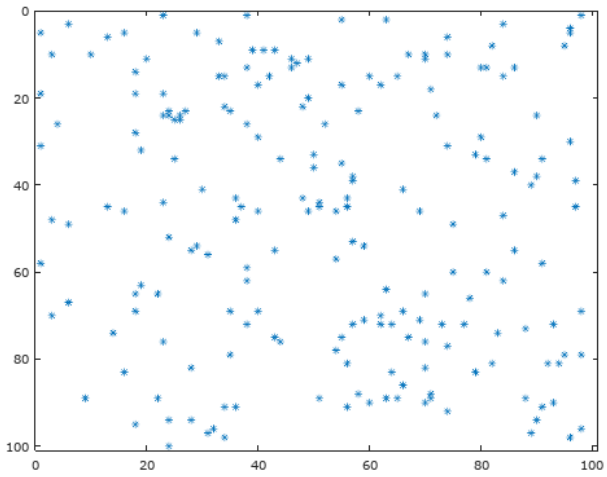


Рисунок 1.4 – Портрет разреженной матрицы

Если в тексте программы встречается оператор input, то выполнение программы приостанавливается, Octave выводит на экран текст подсказки и переходит в режим ожидания ввода. Пользователь вводит с клавиатуры значение и нажимает клавишу Enter. Введенное пользователем значение будет присвоено переменной, имя которой указано слева от знака присваивания.

Одним из основных операторов, предназначенных для ветвления в большинстве языков программирования, является условный оператор. Существует обычная, сокращенная и расширенная формы этого оператора в языке программирования Octave. Обычный условный оператор имеет вид:

if условие
    операторы1
else
    операторы2
end

Здесь условие – логическое выражение, операторы1, операторы2 – операторы языка или встроенные функции Octave. Обычный оператор if работает по следующем алгоритму: если условие истинно, то выполняются операторы1, если ложно – операторы2.

Расширенный условный оператор применяют когда одного условия для принятия решения недостаточно:

if условие1

операторы1
elseif условие2
операторы2
elseif условие 3
операторы3
…
elseif условие n
операторыN
else
операторы
end

Расширенный оператор *if* работает следующим образом. Если условие 1 истинно, то выполняются операторы 1, иначе проверяется условие 2, если оно истинно, то выполняются операторы 2, иначе проверяется условие 3 и т.д. Если ни одно из условий по веткам *elseif* не выполняется, то выполняются операторы по ветке *else*.

Еще одним способом организации разветвлений является оператор альтернативного выбора следующей структуры:

swith параметр
case значение1
операторы1
case значение2
операторы2
case значение3
операторы3
…
otherwise
операторы
end

Оператор *switch* работает следующим образом: если значение параметра равно значениию1, то выполняются операторы1, иначе если параметр равен значениию2, то выполняются операторы2; в противном случае, если значение параметра совпадает со значением3, то выполняются операторы3 и т.д. Если значение параметра не совпадает ни с одним из значений в группах *case*, то выполняются операторы, которые идут после служебного слова *otherwise*.

Конечно, любой алгоритм можно запрограммировать без использования *switch*, используя только *if*, но использование оператора альтернативного

выбора *switch* может сделать программу более компактной.

Оператор цикла с предусловием в языке программирования Octave имеет вид:

while выражение
    операторы
end

Работает цикл с предусловием следующим образом. Вычисляется значение выражения. Если оно истинно, выполняются операторы. В противном случае цикл заканчивается, и управление передается оператору, следующему за телом цикла. Выражение вычисляется перед каждой итерацией цикла. Если при первой проверке выражение ложно, цикл не выполнится ни разу. Выражение должно быть переменной или логическим выражением.

Для записи цикла с известным числом повторений применяют оператор

for параметр = начальное_значение:шаг:конечное_значение
    операторы
end

Выполнение цикла начинается с присвоения параметру цикла начального значения. Затем следует проверка, не превосходит ли параметр цикла конечное значение. Если результат проверки утвердительный, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. В противном случае выполняются операторы в цикле. Далее параметр меняет свое значение на значение шага. Снова производится проверка значения параметра цикла и алгоритм повторяется. Если шаг цикла равен 1, то оператор записывают так

for параметр = начальное значение:конечное значение
    операторы
end

Операторы передачи управления принудительно изменяют порядок выполнения команд. В языке программирования Octave таких операторов два. Операторы *break* и *continue* используют только внутри циклов. Так оператор *break* осуществляет немедленный выход из циклов *while, for* и управление передается оператору, находящемуся непосредственно за циклом. Оператор *continue* начинает новую итерацию цикла, даже если предыдущая не была

завершена.

В языке программирования Octave есть множество функций работы со строками. Рассмотрим некоторые из них.

Таблица 1.4 – Функции для работы со строками

Функция	Описание
char(node)	Возвращает символ по его коду *code*
deblank(s)	Формируется новая строка путем удаления пробелов в конце строки *s*
int2str(x)	Преобразование чисел, хранящихся в массиве (матрице) **x** к целому типу и запись результатов в массив символов
findstr(str,substr)	Возвращает номер позиции, начиная с *str*
sprintf(format, x)	Формирует строку из чисел, хранящихся в числовой переменной **x** в соответствии с форматом *format* (*sprintf 'X=%4.2e',x*)
sscanf(s, format)	Функция возвращает из строки **s** числовое значение или массив значений в соответствии с форматом

В Octave файлы с расширением .m могут содержать не только тексты программ (группа операторов и функций Octave), но и могут быть оформлены как отдельные функции. В этом случае имя функции должно совпадать с именем файла, в котором она хранится (например, функция с именем primer должна храниться в файле primer.m). Функция в Octave имеет следующую структуру. Первая строка функции это заголовок:

function $[y_1,y_2,...y_n]$*=name_function*$(x_1,x_2,...,x_m)$

Здесь *name_function* – имя функции, x_1, x_2, …, x_m – список входных параметров функции, y_1, y_2, …, y_n – список выходных параметров функции. Функция заканчивается служебным словом *end*. Таким образом, в простейшем случае структуру функции можно записать следующим образом:

function $[y_1,y_2,...y_n]$*=name_function*$(x_1,x_2,...,x_m)$
    оператор1;
    оператор2;
    …
    операторm;
end

В файле с расширением .m, кроме основной функции, имя которой совпадает с именем файла, могут находиться так называемые подфункции. Эти

функции доступны только внутри файла. Таким образом, общую структуру функции можно представить так:

```
%Здесь начинается основная функция m-файла,
% имя которой должно совпадать с именем файла, в котором она хранится
function [y1,y2,...yn]=name_function(x1,x2,...,xm)
% Среди операторов основной функции могут быть операторы
% вызова подфункций f1, f2, f3, ...,fl
оператор1;
    оператор2;
    ...
    операторm;
end; % здесь заканчивается основная функция
function [y1,y2,...yn]=f1(x1,x2,...,xm) % начало первой подфункции
    операторы
end % конец первой подфункции
function [y1,y2,...yn]=f2(x1,x2,...,xm) % начало второй подфункции
    операторы
end % конец второй подфункции
...
function [y1,y2,...yn]=fn(x1,x2,...,xm) % начало n-й подфункции
    операторы
end % конец n-й подфункции
```

Такая структура, близка к структуре программ на языке Си. Она не допускает вложенности функций друг в друга. Однако в Octave возможен и другой синтаксис, в котором разрешено использование вложенных функций.

В Octave есть возможность передавать имя функции как входной параметр, что существенно расширяет возможности программирования. Вообще говоря, имя функции передается как строка, а ее вычисление осуществляется с помощью функции feval. Функция feval предоставляет альтернативный способ вычисления значения функции. Параметрами функции feval являются: строка с именем вызываемой функции, в качестве имени может быть встроенная функция или определенная пользователем функция; параметры этой функции, разделенные запятой.

Для измерения времени работы блока кода можно воспользоваться командами tic и toc. Так, происходит измерение времени блока кода, расположенного между строками с этими функциями:

```
tic
    блок кода
toc
```

Иногда необходимо просмотреть информацию по всем переменными или по какой-либо определенной переменной. Это можно сделать через область переменных (см. рисунок 1.1) или воспользоваться командой who или who(), которая выводит список определенных переменных, или командой whos или whos() – список переменных с указанием их размера и объема занимаемой памяти. Следующий пример иллюстрируют действие этих команд.

```
>> A=[1 2; 3 4];
>> b='line';
>> c=5;
>> d=1+i;
>> who
Variables in the current scope:
A  b  c  d
>> whos
Variables in the current scope:
   Attr      Name      Size                Bytes     Class
   ====      ====      ====                =====     =====
               A       2x2                    32     double
               b       1x4                     4     char
               c       1x1                     8     double
     c         d       1x1                    16     double
Total is 10 elements using 60 bytes
>> whos d
Variables in the current scope:
   Attr      Name      Size                Bytes     Class
   ====      ====      ====                =====     =====
     c         d       1x1                    16     double
Total is 1 element using 16 bytes
```

Представленное изложение функции Octave не является исчерпывающим, но даёт общее представление о них и их использовании в рамках данного курса. По мере необходимости будут использованы и другие функции, неописанные выше, с соответствующими комментариями.

1.1.2 Scilab

Scilab (сайлаб) еще один свободно распространяемый язык

программирования высокого уровня система, также как и Octave, предназначенный для выполнения инженерных и научных вычислений. Она является еще одной бесплатной альтернативой пакету MATLAB. Синтаксис языка в большей степени совпадает с синтаксисом Octave, но имеются и некоторые отличия. Рассмотрим основные из них.

На рисунке 1.5 приведено главное окно Scilab (ver. 5.5.2). Видно, что и в нем есть те же области, что и у Octave, но их названия отличаются между собой. Для перехода в редактор нужно в панели инструментов выбрать Инструменты и далее текстовый редактор SciNotes, после чего можно перейти к программированию. Файлы с программой имеют расширение .sce. Для вывода нужно использовать функцию disp(аргумент), где в качестве аргумента может быть скаляр, матрица, строка и тп. (disp([1 2],3) , disp("a",1,"c")) Однако при работе в командном окне достаточно не ставить «;» в конце строки для вывода на печать. Для комментариев используется символ «//», а не «%».

это свободно распространяемый язык программирования высокого уровня, ориентированный на проведение численных расчетов, и, по сути, являющийся альтернативой коммерческому пакету MATLAB

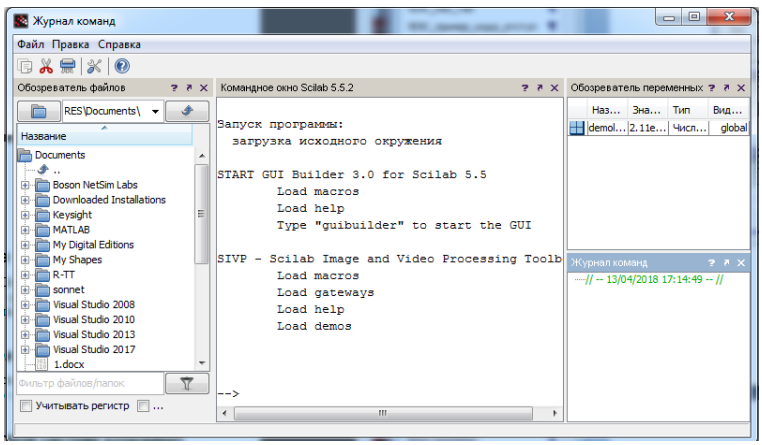


Рисунок 1.5 – Главное окно Scilab

Для обращения к системным переменным, кроме ans, перед их имен нужно вставить «%». Для мнимой единицы зарезервировано только имя i. Рассмотрим пример.

-->x=%pi

```
 x  =
      3.1415927
-->x=x+%i
 x  =
      3.1415927 + i
```

Также имеются различия в задании формата значений, выводимых на экран. Это все та же команда format, но с другим синтаксисом: format([type],[num]), где type символьная строка ("v" для переменного формата (по умолчанию) или "e" для формата экспоненциальной записи), а num – целое число от 2 до 25 (по умолчанию равно 10), определяющее количество знаков. Функция format() возвращает вектор текущего формата: первый элемент – это тип формата (1, если "v" и 0, если "e"); второй элемент – это число знаков. Таким образом, format long в Octave соответствует format(17) в Scilab, а format short соответственно format(7). Рассмотрим пример.

```
-->%pi
 %pi  =
      3.1415927
-->format(25);%pi
 %pi  =
      3.1415926535897931159980
-->format('e',15);%pi
 %pi  =
      3.14159265D+00
-->format()
ans  =
      0.00000000D+00    1.50000000D+01
```

Для измерения времени используются не команды, а функции tic() и toc(). Как и в Octave происходит замер времени выполнения блока кода, расположенного между ними. Еще одним отличием является то, что в Scilab для вывода затраченного времени требуется дополнительно воспользоваться функцией disp(). Продемонстрируем на примере небольшой программы для поэлементного возведения в квадрат каждого элемента матрицы.

```
tic()
a=[1 2; 3 4];
a=a.^2;
t=toc()
disp (t)
```

Для просмотра информации по конкретной переменной в Scilab используется конструкция whos –name d в отличие от Octave, где используется вместо whos d. Также может быть использована команда who_user() для просмотра только пользовательских переменных.

1.1.3 Вычисление градиента, дивергенции и ротора

В данном подразделе приведены примеры вычисления градиента, ротора и дивергенции, в пакете Octave. Сначала приведены постановки задач, примеры реализации, а затем задания для самостоятельного выполнения.

1.1.3.1 Вычисление градиента

Проиллюстрируем особенности вычисления градиента на примере. Пусть дана поверхность $z = \exp(-(x^2+y^2))$. Необходимо найти градиент ∇z аналитически и вычислить его в точке ($x = 0$, $y = 0$). Далее используя Octave или Scilab, построить профиль этой поверхности, контуры равных значений функции z (линии уровня) и начертить поле ∇z. Графическое построение проводить в интервалах $x \in [-2; 2]$, $y \in [-2; 2]$ с шагом по осям x и y равным 0,2. Графики вывести в одном окне.

Решение. $\nabla z = -2\,x \exp(-(x^2+y^2))\mathbf{x}_0 + -2\,y \exp(-(x^2+y^2))\mathbf{y}_0$. $\nabla z = 0$ в точке (x=0, y=0). Продемонстрируем программный код Octave для решения задачи с сопутствующими комментариями. Для создания рисунков использованы функции contour и quiver. Далее приведен код результирующей программы:

```
clear all;
clc;
clf;
surf_min=-2;
surf_max=2;
surf_step=0.2;
gradient_step = 0.2;
%сетка для построения функции
[x,y]=meshgrid(surf_min:surf_step:surf_max);
%сетка для построения градиента
[xx,yy]=meshgrid(surf_min:gradient_step:surf_max);
z=exp(-x.^2-y.^2);
```

```
zz=exp(-xx.^2-yy.^2);
[u,v]=gradient(zz,gradient_step);
subplot(1,2,1)
surf(x,y,z);
axis square; %одинаковый диапазон изменения переменных по осям.
view(-30,20);
subplot(1,2,2)
contour(x,y,z,5);
hold on
axis square;
%axis equal; %масштаб, который обеспечивает одинаковые
расстояния между метками по осям x и y
quiver(xx,yy,u,v)
hold off
%Тестирование (сравнение вычислений и аналитики)
test_indexes=[7,8];
test_x=xx(test_indexes(1),test_indexes(2));
test_y=yy(test_indexes(1),test_indexes(2));
gradient_x=u(test_indexes(1),test_indexes(2));
gradient_y=v(test_indexes(1),test_indexes(2));
%значения, полученные аналитически
%grad z = -2xexp[-(x^2+y^2)]x0 - 2yexp[-(x^2+y^2)]y0
analitic_x = -2*test_x*exp(-(test_x^2+test_y^2))
analitic_y =- 2*test_y*exp(-(test_x^2+test_y^2))
fprintf('x=%.10f y=%.10f\n',test_x,test_y);
fprintf('analitic_x=%.10f analitic_y=%.10f\n',analitic_x,analitic_y);
fprintf('gradient_x=%.10f gradient_y=%.10f\n',gradient_x,gradient_y);
```

Задание. Проработать описанный программный код, построить профиль, контуры равных значений функции z (линии уровня) и получить поле ∇z для поверхности $z = -2\exp(-0{,}1(x^2+y^2))$.

1.1.3.2 Вычисление дивергенции

Дано векторное поле в цилиндрической системе координат $\mathbf{a} = \rho \exp(-(\rho/\alpha)^2)\boldsymbol{\rho}_0$, $\alpha=3$. Необходимо найти аналитически и, используя Octave, дивергенцию градиент $\nabla\cdot\mathbf{a}$; построить векторное поле **a** и контуры равных значений дивергенции. Графическое построение проводить в интервалах $x\in[-2; 2]$, $y\in[-2; 2]$ с шагом по осям x и y равным 0,5. Графики выводить в одном окне.

Решение. В цилиндрической системе координат: $\nabla \cdot \mathbf{a} = \frac{1}{\rho}\frac{\partial(\rho\, a_\rho)}{\partial \rho} + \frac{1}{\rho}\frac{\partial a_\varphi}{\partial \varphi} + \frac{\partial a_z}{\partial z}$. Тогда

$$\nabla \cdot \mathbf{a} = \frac{1}{\rho}\frac{\partial(\rho\, a_\rho)}{\partial \rho} = \frac{1}{\rho}\frac{\partial}{\partial \rho}(\rho^2 \exp(-(\rho/\alpha)^2) = 2\exp(-(\rho/\alpha)^2)[1-\frac{\rho^2}{\alpha^2}].$$

Для вычисления дивергенции $\nabla\cdot\mathbf{a}$ необходимо использовать команду divergence (при этом необходимо перейти в декартову систему координат, учитывая, что $\rho = \sqrt{x^2 + y^2}$), а для создания рисунков – команды quiver и contour. Далее приведен итоговый код программы.

```
clear all;
clc;
clf;
alpha=3;
mesh_min=-2;
mesh_max=2;
mesh_step=0.5;
[x,y]=meshgrid(mesh_min:mesh_step:mesh_max,mesh_min:mesh_step:mesh_m
ax);
r2=x.^2+y.^2;
Ax=exp(-r2./alpha.^2).*x;
Ay=exp(-r2./alpha.^2).*y;
D=divergence(x,y,Ax,Ay);
subplot(2,1,1)
quiver(x,y,Ax,Ay);
axis square
subplot(2,1,2)
hold on
axis square
contour(x,y,D,5)
hold off
%Тестирование (сравнение вычислений и аналитики)
test_indexes=[7,8];
test_x=x(test_indexes(1),test_indexes(2));
test_y=y(test_indexes(1),test_indexes(2));
test_r2=test_x^2+test_y^2;
%Значение дивергенции, расчитанное с помощью функции divergence
test_div=D(test_indexes(1),test_indexes(2));
%Значение дивергенции, рассчитанное аналитически
analitical_div=2*exp(-test_r2/(alpha^2))*(1-test_r2/(alpha^2));
```

```
fprintf('x=%.5f y=%.5f\n',test_x,test_y);
fprintf('analitic_div=%.5f\n',analitical_div);
fprintf('test_div=%.5f\n',test_div);
```

Задание. Проработать программный код и найти дивергенцию $\nabla\cdot\mathbf{a}$; построить векторное поле $\mathbf{a}$ и контуры равных значений дивергенции для поля $\mathbf{a} = 3\rho \exp(-0{,}5(\rho/\alpha)^2)\boldsymbol{\rho}_0$, $\alpha=2$.

1.1.3.3 Вычисление ротора

Задано векторное поле в цилиндрической системе координат $\mathbf{a} = \beta\rho \exp(-(\rho/\alpha)^2)\boldsymbol{\varphi}_0$, $\alpha=3$, $\beta=1$. Необходимо найти аналитически и, используя Octave (при этом необходимо перейти в декартову систему координат, учитывая, что $\rho = \sqrt{x^2 + y^2}$), ротор $\nabla\times\mathbf{a}$; построить векторное поле $\nabla\times\mathbf{a}$ и контуры равных значений z-компоненты ротора $\nabla\times\mathbf{a}$. Графическое построение проводить в интервалах $x\in[-2; 2]$, $y\in[-2; 2]$ с шагом по осям x и y равным 0,2. Графики выводить в одном окне.

Решение. В цилиндрической системе координат:

$$\nabla\times\mathbf{a} = \frac{1}{\rho}\begin{vmatrix} \boldsymbol{\rho}_0 & \rho\boldsymbol{\varphi}_0 & \mathbf{z}_0 \\ \partial/\partial\rho & \partial/\partial\varphi & \partial/\partial z \\ a_\rho & \rho a_\varphi & a_z \end{vmatrix} = 2\beta\exp(-(\rho/\alpha)^2)[1-\frac{\rho^2}{\alpha^2}]\mathbf{z}_0.$$

Для вычисления ротора используется команда curl, а для отображения – команды quiver и contour. Итоговый программный код программы приведен ниже.

```
clear all;
clc;
clf;
alpha=3;
beta=1;
mesh_min=-2;
mesh_max=2;
mesh_step=0.2;
[x,y]=meshgrid(mesh_min:mesh_step:mesh_max,mesh_min:mesh_step:mesh_max);
r2=x.^2+y.^2;
```

```
Ax=-exp(-r2./alpha.^2).*y*beta;
Ay=exp(-r2./alpha.^2).*x*beta;
C=curl(x,y,Ax,Ay);
subplot(1,2,1)
quiver(x,y,Ax,Ay);
axis square
subplot(1,2,2)
hold on
axis square
contour(x,y,C,5)
hold off
%Тестирование (сравнение вычислений и аналитики)
test_indexes=[7,8];
test_x=x(test_indexes(1),test_indexes(2));
test_y=y(test_indexes(1),test_indexes(2));
test_r2=test_x^2+test_y^2;
%Значение ротора, расчитанное с помощью функции curl
test_curl=C(test_indexes(1),test_indexes(2));
%Значение ротора, рассчитанное аналитически
analitical_curl=2*beta*exp(-test_r2/(alpha^2))*(1-test_r2/(alpha^2));
fprintf('x=%.5f y=%.5f\n',test_x,test_y);
fprintf('analitic_div=%.5f\n',analitical_curl);
fprintf('test_div=%.5f\n',test_curl);
```

Задание. Проработать программный код и найти ротор $\nabla\times\mathbf{a}$; построить векторное поле $\nabla\times\mathbf{a}$ и контуры равных значений z-компоненты ротора $\nabla\times\mathbf{a}$ для поля $\mathbf{a} = \beta\rho\exp(-1{,}5(\rho/\alpha)^2)\boldsymbol{\varphi}_0$, $\alpha=1$, $\beta=3$.

1.2 Методы решения СЛАУ

1.2.1 Прямые методы

1.2.1.1 LU-разложение

Эффективным методом решения системы линейных алгебраических уравнений является метод разложения на треугольные матрицы, или LU-разложение. Алгоритмы этого метода близки к методу исключения Гаусса, хотя вычисления могут производится в различной последовательности. LU-разложение матрицы **A** можно представить в виде **A**=**LU**, где **L** – нижнетреугольная матрица с единичной диагональю, а **U** – верхнетреугольная матрица. При этом решение СЛАУ, при известном LU-

разложении, сводится к решению системы:

$$\begin{cases} \mathbf{Ux} = \mathbf{y} \\ \mathbf{Ly} = \mathbf{b} \end{cases}. \qquad (1.1)$$

Вначале находится LU-разложение матрицы, затем решается второе уравнение, а затем первое системы (1.1).

Существует порядка десяти вариантов реализации разложения. Чаще всего используются, так называемых *ijk*-формы со следующим обозначением индексов:

- k – номер исключаемой переменной;
- i – номер строки, т.е. модифицируемого уравнения;
- j – номер столбца, т.е. коэффициента в модифицируемом уравнении.

Тогда общую основу всех алгоритмов удобно определить тройкой вложенных циклов вида.

Для___
  Для___
    Для___
      $a_{ij} = a_{ij} - l_{ik}a_{kj}$

Далее приведены используемые в данной работе *ijk*-алгоритмы.

LU-разложение, *ikj*-алгоритм

Для $i = 2, \ldots, N$
  Для $k = 1, \ldots, i - 1$
    $a_{i,k} = a_{i,k} / a_{k,k}$
      Для $j = k + 1, \ldots, N$
        $a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
      Увеличить j
  Увеличить k
Увеличить i

Данный алгоритм позволяет пересчитать i-ю строку матрицы **A** в i-ю строку матриц **L** и **U**. Каждые 1, 2,…, $j - 1$ строки участвуют в определении j-й строки матриц **L** и **U**, но сами больше не модернизируются. Таким образом, доступ к элементам матрицы A производится по строкам. Исключение по строкам. Модификации отложенные.

LU-разложение, *ijk*-алгоритм

Для $i = 2, \ldots, N$
  Для $j = 2, \ldots, i$
    $a_{i,j-1} = a_{i,j-1} / a_{j-1,j-1}$
      Для $k = 1, \ldots, j-1$
        $a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
      Увеличить k

Доступ к элементам матрицы **A** производится по строкам. Исключение по строкам. Модификации отложенные. Первый цикл по j элементы i-й строки матрицы **L**. Второй цикл по j

Увеличить *j*
Для *j* = *i*+1, …, *N*
Для *k* = 1, …, *i*–1
$a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
Увеличить *k*
Увеличить *j*
Увеличить *i*

элементы *i*-й строки **U**.

LU-разложение, *kij*-алгоритм

Для *k* = 1, …, *N* – 1
Для *i* = *k* + 1, …, *N*
$a_{i,k} = a_{i,k} / a_{k,k}$
Для *j* = *k* + 1, …, *N*
$a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
Увеличить *j*
Увеличить *i*
Увеличить *k*

Доступ к элементам матрицы **A** производится по строкам. Исключение по столбцам. Модификации немедленные.

LU-разложение, *kji*-алгоритм

Для *k* = 1, …, *N* – 1
Для *s* = *k* + 1, …, *N*
$a_{s,k} = a_{s,k} / a_{k,k}$
Увеличить s
Для *j* = *k* + 1, …, *N*
Для *i* = *k* + 1, …, *N*
$a_{i,j} = a_{i,j} - a_{i,k} \cdot a_{k,j}$
Увеличить *i*
Увеличить *j*
Увеличить *k*

Доступ к элементам матрицы **A** производится по столбцам. Исключение по столбцам. Модификации немедленные.

В ходе работы требуется реализовать представленные алгоритмы в виде функций. Пример реализации *ikj*-алгоритма в виде функции B=LUikj(A):

```
function [B]=LUijk(A)
B=A;
m=size(A);
for i = 2:m
   for k = 1:(i-1)
     B(i,k) = B(i,k) / B(k,k);
      for j=(k+1):m
        B(i,j) = B(i,j) - B(i,k)*B(k,j);
      end
   end
end
end
```

Далее с помощью реализованных функций получить время, затрачиваемое на LU-разложение, для произвольно заданной матрицы порядка 100, 200, …, 1500. Построить полученные зависимости. Пояснить получены результаты, в том числе с помощью оценки вычислительной сложности реализованных функций с использованием *O*-нотации.

1.2.1.2 Решение систем с треугольными матрицами

С помощью одной из функций, реализованных ранее для получения LU-разложения, решить СЛАУ вида

$$\begin{cases} 1{,}83x_1 + 4{,}34x_2 - 7{,}49x_3 + 11{,}07x_4 = 1 \\ -2{,}15x_1 + 4{,}94x_2 - 3{,}89x_3 + 6{,}48x_4 = 2 \\ -0{,}50x_1 + 1{,}94x_2 - 3{,}32x_3 + 4{,}33x_4 = 3 \\ -4{,}27x_1 + 8{,}45x_2 - 7{,}71x_3 + 12{,}30x_4 = 4 \end{cases}.$$

После получения LU-разложения требуется реализовать 2 функции для решения системы вида (1.1). Данный процесс состоит из решения двух уравнений с треугольными матрицами, сначала с **L**, а затем с **U**. Далее приведен пример функции (*backsubL*()) для решения уравнения вида **Ly** = **b**:

```
function Y = backsubL(A, B)
n = length(B);
Y = zeros(n, 1);
Y(1) = B(1);
for  k = 2:n
   Y(k) = (B(k)-A(k,k-1:-1:1)*Y(k-1:-1:1,1));
end
```

Необходимо проработать программную реализацию. После чего реализовать аналогичную функцию для решения уравнения вида **Ux** = **y** с именем *backsubU*() и проверить правильность реализации на примере приведенной выше СЛАУ с помощью стандартных средств Octave (Scilab). Оценить вычислительную сложность реализованных функций с помощью *O*-нотации.

1.2.1.3 Обращение матриц

С помощью функций, реализованных на предыдущих занятиях, написать программу для нахождения обратной матрицы с помощью LU-разложения. Правильность реализации проверить с помощью стандартных средств Octave (Scilab). Оценить вычислительную сложность реализации с помощью *O*-нотации. Сравнить с оценками, полученными на предыдущих занятиях для последовательного LU-разложения и решения двух уравнений с треугольными матрицами (система (1.1)). Пояснить результаты.

1.2.1.4 Особенности программной реализации

В ходе занятия требуется проанализировать назначение следующих функций.

```
function RESULT = unknown_func1(A)
[n, n] = size(A);
for k = 1:n-1
   rows = k+1:n;
   A(rows, k) = A(rows, k)/A(k, k);
   A(rows, rows) = A(rows, rows) - A(rows, k) * A(k, rows);
end
RESULT = A;

function RESULT = unknown_func2(A)
[n, n] = size(A);
for k = 1:n-1
   A(k+1:n, k) = A(k+1:n, k) / A(k, k);
   for i = k+1:n
      A(i, k+1:n) = A(i, k+1:n) - A(i, k) * A(k, k+1:n);
   end
end
RESULT = A;

function RESULT = unknown_func3(A)
[n, n] = size(A);
for k = 1:n-1
   A(k+1:n, k) = A(k+1:n, k) / A(k, k);
   for i = k+1:n
      for j = k+1:n
         A(i, j) = A(i, j) - A(i, k) * A(k, j);
```

```
      end
    end
end
RESULT = A;
```

Далее оценить их вычислительную сложность с помощью *O*-нотации и время, затрачиваемое на их выполнение, для произвольно заданной матрицы порядка 100, 200, …, 1500. Построить график с помощью полученных данных и пояснить результаты.

1.2.1.5 Блочное LU-разложение

Изменить имя функции последовательного LU-разложения *unknown_func1*() на *lu_sequential*(). Затем проанализировать, как работает следующая функция, где bl – размер блока, $1 \le bl \le N$, N – порядок матрицы.

```
function RESULT = lu_block(A, bl)
[N, N] = size(A);
B = zeros(bl, N);
C = zeros(N, bl);
for l = 1:bl:N
  r = min(N, l + bl - 1);
  A(l:r,l:r) = lu_sequential(A(l:r,l:r));
  if (r == N)
    break;
  end
  B(1, r+1:N) = A(l, r+1:N);
  for k = 1:r-l
     B(k+1,r+1:N) = A(l+k, r+1:N) - ...
     A(l+k, l:l+k-1)*B(1:k, r+1:N);
  end
  A(l:r, r+1:N) = B(1:r-l+1, r+1:N);
  C(r+1:N, 1) = A(r+1:N, l) / A(l, l);
  for k = 1:r-l
     C(r+1:N, k+1) = (A(r+1:N, l+k) - ...
     C(r+1:N, 1:k)*A(l:l+k-1, l+k)) / A(l+k, l+k);
  end
  A(r+1:N, l:r) = C(r+1:N, 1:r-l+1);
  A(r+1:N, r+1:N) = A(r+1:N, r+1:N) - ...
  C(r+1:N, 1:r-l+1)*B(1:r-l+1, r+1:N);
end
RESULT = A;
```

На примере произвольной матрицы порядка 10 протестировать работу функций *lu_sequential*() и *lu_block*() при разном размере блока. Проанализировать полученные результаты. Оценить вычислительную сложность алгоритма блочного LU-разложения и сравнить с полученными ранее оценками для последовательного разложения. Оценить время работы обеих функций на примере произвольно заданной матрицы порядка 1000 и пояснить полученные результаты. Затем подобрать оптимальный размер блока для функции, реализующей блочное LU-разложение, с точки зрения минимизации временных затрат.

1.2.1.6 Разложение Холецкого

Рассмотрим процесс решения СЛАУ с симметричной матрицей **A**. Для их решения часто применяется метод Холецкого. В случае, если **A** еще и положительно определенная матрица, данный частный случай чаще называют методом квадратных корней.

В основе метода лежит алгоритм специального LU-разложения матрицы **A**, в результате чего она приводится к виду $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, где нижняя треугольная матрица

$$\mathbf{L} = \begin{pmatrix} l_{11} & 0 & 0 & \cdots & 0 & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ l_{N1} & l_{N2} & l_{N3} & \cdots & l_{NN-1} & l_{NN} \end{pmatrix}$$

уже не обязательно должна иметь на главной диагонали единицы, как это было в методе Гаусса, а требуется только, чтобы диагональные элементы l_{ii} были положительными. Если такое разложение получено, то решение системы $\mathbf{Ax} = \mathbf{b}$ сводится к последовательному решению двух систем с треугольными матрицами:

$$\mathbf{Ly} = \mathbf{b},\ \mathbf{L}^T\mathbf{x} = \mathbf{y}. \tag{1.1}$$

Для решения требуется выполнение порядка $2N^2$ арифметических операций.

Далее приведена функция, осуществляющая разложение Холецкого.

```
function RESULT = CHOL(A)
 [N, N] = size(A);
for k = 1:N
   A(k,k) = sqrt(A(k,k));
   A(k+1:N,k) = A(k+1:N,k) / A(k, k);
   for j = k+1:N
      A(j:N,j) = A(j:N,j) - A(j:N,k)*A(j,k);
   end
end
RESULT = tril(A);
```

В ходе работы требуется разобрать работы данной функции и реализовать с её помощью алгоритм решения СЛАУ. Правильность работы продемонстрировать на примере СЛАУ

$$\begin{aligned} 6{,}25x_1 &- x_2 + 0{,}5x_3 = 7{,}5 \\ -x_1 &+ 5x_2 + 2{,}12x_3 = -8{,}68\,. \\ 0{,}5x_1 &+ 2{,}12x_2 + 3{,}6x_3 = -0{,}24 \end{aligned}$$

1.2.2 Итерационные методы

1.2.2.1 Метод Якоби

Метод Гаусса, как и методы, основанные на LU- и QR-разложении, относятся к прямым методам решения СЛАУ, т.е. если отбросить ошибку округления, решение, полученное посредством данных методов, является точным. Другую группу составляют итерационные методы, при помощи которых решение находится путем последовательных приближений. Если итерационный процесс сходится, то каждое последующее приближение уточняет предыдущее. Представим матрицу системы **Ax** = **b** в виде:

$$\mathbf{A} = \mathbf{L} + \mathbf{P} + \mathbf{D}, \tag{1.2}$$

где

$$\mathbf{D} = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 & 0 \\ 0 & a_{22} & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 0 & a_{nn} \end{pmatrix}, \mathbf{L} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ a_{21} & 0 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{n,n-1} & a_{nn} \end{pmatrix},$$

$$\mathbf{P}=\begin{pmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1,n-1} & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2,n-1} & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

Предположим, что $a_{ii} \neq 0$, $i = 1, 2, \ldots, n$. Тогда систему $\mathbf{Ax} = \mathbf{b}$, с учетом (1.2), можно записать в виде:

$$\mathbf{Lx} + \mathbf{Dx} + \mathbf{Px} = \mathbf{b},$$

или

$$\mathbf{x} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{P})\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}. \qquad (1.3)$$

Сравнивая $\mathbf{Ax} = \mathbf{b}$ и (1.3), видно, что

$$\mathbf{B} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{P}),\ \mathbf{C} = \mathbf{D}^{-1}.$$

На основе формулы (1.3) записывается итерационный процесс метода Якоби:

$$\mathbf{x}_{k+1} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{P})\mathbf{x}_k + \mathbf{D}^{-1}\mathbf{b}. \qquad (1.4)$$

Пусть $\mathbf{x}_k = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right)$. Тогда матричная формула (1.4) с учетом вида матриц **L**, **P** и $\mathbf{D}^{-1}$ записывается покомпонентно в виде:

$$\begin{cases} x_1^{(k+1)} = \dfrac{b_1}{a_{11}} - \dfrac{a_{12}}{a_{11}} x_2^{(k)} - \dfrac{a_{13}}{a_{11}} x_3^{(k)} - \ldots - \dfrac{a_{1n}}{a_{11}} x_n^{(k)} \\ x_2^{(k+1)} = \dfrac{b_2}{a_{22}} - \dfrac{a_{21}}{a_{22}} x_2^{(k)} - \dfrac{a_{23}}{a_{22}} x_3^{(k)} - \ldots - \dfrac{a_{2n}}{a_{22}} x_n^{(k)} \\ \cdots \\ x_n^{(k+1)} = \dfrac{b_n}{a_{nn}} - \dfrac{a_{n1}}{a_{nn}} x_2^{(k)} - \dfrac{a_{n3}}{a_{nn}} x_3^{(k)} - \ldots - \dfrac{a_{n,n-1}}{a_{nn}} x_n^{(k)} \end{cases},$$

или

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)},\ i = 1,2,\ldots,n. \qquad (1.5)$$

Тогда матрица **B** имеет вид:

$$\mathbf{B}=\begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} & \dots & -\frac{a_{1,n-1}}{a_{11}} & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & \dots & -\frac{a_{2,n-1}}{a_{22}} & -\frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & -\frac{a_{n3}}{a_{nn}} & \dots & -\frac{a_{n,n-1}}{a_{nn}} & 0 \end{pmatrix}.$$

Для сходимости метода Якоби достаточно, чтобы $\|\mathbf{B}\| < 1$. Достаточном условием сходимости метода Якоби является:

$$\sum_{\substack{j=1 \\ i\neq j}}^{n} |a_{ij}| < |a_{ii}|,\ i=1,2,\dots,n. \qquad (1.6)$$

Неравенство (1.6) означает диагональное преобладание в исходной матрице **A**, которого следует добиться до вычислений согласно формуле (1.5). Рассмотрим пример. Пусть дана система уравнений:

$$\begin{cases} 100x_1 + 30x_2 - 70x_3 = 60 \\ 15x_1 - 50x_2 - 5x_3 = -40 \\ 6x_1 + 2x_2 + 20x_3 = 28 \end{cases},$$

где

$$\mathbf{A}=\begin{pmatrix} 100 & 30 & -70 \\ 15 & -50 & -5 \\ 6 & 2 & 20 \end{pmatrix}, \mathbf{b}=\begin{pmatrix} 60 \\ -40 \\ 28 \end{pmatrix}.$$

В исходной матрице **A** диагонального преобладания нет. Поэтому до основных вычислений выполним преобразования. Сложим первые два уравнения:

$$\begin{cases} 115x_1 - 20x_2 - 75x_3 = 20 \\ 15x_1 - 50x_2 - 5x_3 = -40 \\ 6x_1 + 2x_2 + 20x_3 = 28 \end{cases}.$$

В результате матрица СЛАУ преобразуется к матрице с диагональным преобладанием. Тогда итерационный процесс можно записать следующим образом:

$$\begin{cases} x_1^{(k+1)} = \dfrac{20}{115} + \dfrac{20}{115} x_2^{(k)} + \dfrac{75}{115} x_3^{k} \\ x_2^{(k+1)} = \dfrac{40}{50} + \dfrac{15}{50} x_1^{(k)} - \dfrac{5}{50} x_3^{(k)} \\ x_3^{(k+1)} = \dfrac{28}{20} - \dfrac{6}{20} x_1^{(k)} - \dfrac{2}{20} x_2^{(k)} \end{cases}.$$

При этом матрица **B** примет вид:

$$\mathbf{B} = \begin{pmatrix} 0 & 0{,}17391 & 0{,}65217 \\ 0{,}3 & 0 & -0{,}1 \\ -0{,}3 & -0{,}1 & 0 \end{pmatrix}.$$

Нормы матрицы $\|\mathbf{B}\|_1 = 0{,}826 < 1$, $\|\mathbf{B}\|_2 = 0{,}75217 < 1$, что говорит о выполнении условий сходимости метода Якоби. В качестве начального приближения возьмем:

$$\mathbf{x}_0 = \mathbf{C} \cdot \mathbf{b} = \begin{pmatrix} 0{,}17391 \\ 0{,}8 \\ 1{,}4 \end{pmatrix}.$$

Результат вычисления 6 итераций приведен ниже:

$$\begin{cases} x_1^{(1)} = 1{,}2260 \\ x_2^{(1)} = 0{,}7122 \\ x_3^{(1)} = 1{,}2678 \end{cases}, \begin{cases} x_1^{(2)} = 1{,}2246 \\ x_2^{(2)} = 1{,}041 \\ x_3^{(2)} = 0{,}961 \end{cases}, \begin{cases} x_1^{(3)} = 0{,}9817 \\ x_2^{(3)} = 1{,}0413 \\ x_3^{(3)} = 1{,}2829 \end{cases},$$

$$\begin{cases} x_1^{(4)} = 1{,}1917 \\ x_2^{(4)} = 0{,}9662 \\ x_3^{(4)} = 1{,}0014 \end{cases}, \begin{cases} x_1^{(5)} = 0{,}9950 \\ x_2^{(5)} = 1{,}0574 \\ x_3^{(5)} = 0{,}9459 \end{cases}, \begin{cases} x_1^{(6)} = 0{,}9747 \\ x_2^{(6)} = 1{,}0039 \\ x_3^{(6)} = 0{,}9958 \end{cases}.$$

Из полученных данных хорошо видна сходимость от итерации к итерации.

Требуется программно реализовать метод Якоби для решения СЛАУ. Для проверки правильности реализации использовать приведенный выше пример.

1.2.2.2 Метод Гаусса-Зейделя

Для реализации метода Гаусса-Зейделя модифицировать разработанную на предыдущем занятии программу. Выполнить сравнение с результатами, полученными с помощью метода Якоби.

1.3 Метод конечных разностей

1.3.1 Прямоугольная область: явная схема

Найти распределение потенциала в прямоугольной области с помощью метод конечных разностей, пятиточечного шаблона и метод простых итераций (явная схема). Область ограничена идеально проводящими металлическим электродами (рисунок 1.9). Наличие небольших зазоров между электродами позволяет поддерживать между ними отличную от нуля разность потенциалов. Величина на каждом электроде постоянна и не зависит от времени. Пространство разбивается на 12 подобластей, в результате образовано шесть внутренних узлов с номерами от I до VI и 14 внешних узлов (от *a* до *n*), потенциал которых совпадает с потенциалом соответствующей границы.

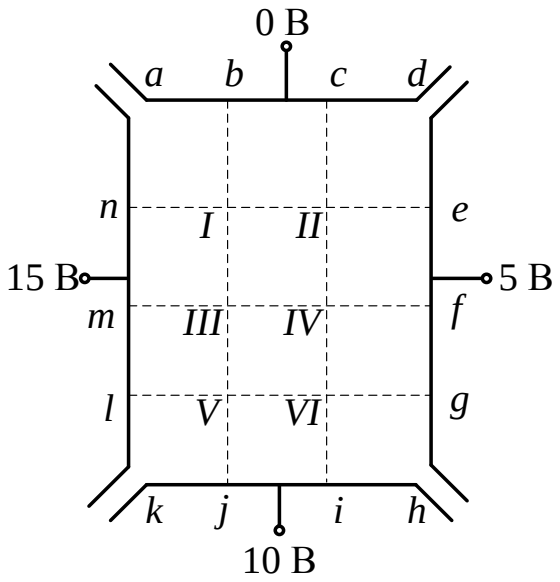


Рисунок 1.9 – Исследуемая структура

Далее приведен программный код возможной реализации.

```
clc;
clear;
ny=5;
nx=4;
V=zeros(ny,nx);
ITRY =100;
V(:,1)=15;
V(:,4)=5;
V(5,:)=10;
```

```
tol=0.01;
old=zeros(3,2);
for iter = 1:ITRY
   for i = 2:nx-1
      for j = 2:ny-1
         V(j,i) = (V(j,i+1)+V(j,i-1)+V(j+1,i)+V(j-1,i))/4;
      end
   end
   new=V(2:4,2:3);
   if norm(new-old)/norm(new)<tol
      iter
      V(2:4,2:3)
      break;
   end
   old=new;
end
```

Задание. Разобрать работу представленного программного кода и разработать программу для анализа данной области при разбиении пространства на 36 подобластей.

1.3.2 Прямоугольная область: явная схема, ускорение сходимости

Из анализа предыдущей реализации видно, что текущее значение потенциала никак не использует информацию о его значении на предыдущей итерации. Поэтому целесообразно перейти от использования метода простой итерации, например, к методам релаксации. В однородной двухмерной среде получим:

$$\varphi_{ij}^{новое} = R\left(\frac{\varphi_{i+1\,j} + \varphi_{i-1\,j} + \varphi_{ij+1} + \varphi_{ij-1}}{4} + (1-R)\,\varphi_{ij}^{старое}\right).$$

Здесь R – параметр релаксации. При $R \le 1$ получим метод нижней релаксации, при $R > 1$ – метод верхней релаксации. Хотя не существует единого оптимального значения множителя R для всех классов задач, описываемых конечно-разностными уравнениями, обычно при R = 1,5 достигается достаточно быстрая сходимость, что приводит к эффективной реализации для широкого круга задач. Далее приведен соответствующий программный код.

```
clc;
clear;
```

```
ny=5;
nx=4;
V=zeros(ny,nx);
 ITRY =100;
 Res=1.5;
 V(:,1)=15;
 V(:,4)=5;
 V(5,:)=10;
 tol=0.1;
 old=zeros(3,2);
for iter = 1:ITRY
   for i = 2:nx-1
      for j = 2:ny-1
         V(j,i) = Res*((V(j,i+1)+V(j,i-1)+V(j+1,i)+V(j-1,i))/4)+((1-Res)*V(j,i));
      end
   end
   new=V(2:4,2:3)
   norm_new=norm(new);
    if norm(new-old)/norm(new)<tol
      iter
      V(2:4,2:3)
      break;
   end
   old=new;
end
```

Задание. Разобрать работу представленного программного кода и выполнить сравнение сходимости при разных значениях параметра релаксации (взять 5–6 значений параметра *R*).

1.3.3 Прямоугольная область: неявная схема

Альтернативой методам простой итерации и релаксации может служить матричный подход (неявная схема), рассматриваемый далее. Так, требуется решить предыдущий пример с использованием неявной схемы (матричного уравнения). Для этого для каждого узла запишем конечно-разностное уравнение, всего их для данной задачи будет шесть:

$$4V_{\mathrm{I}} = V_b + V_{\mathrm{II}} + V_{\mathrm{III}} + V_n,$$

$$4V_{\mathrm{II}} = V_c + V_e + V_{\mathrm{IV}} + V_{\mathrm{I}},$$

$$4V_{\mathrm{II}} = V_{\mathrm{I}} + V_{\mathrm{IV}} + V_{\mathrm{V}} + V_m,$$

$$4V_{IV} = V_{II} + V_f + V_{VI} + V_{III},$$
$$4V_V = V_{III} + V_{VI} + V_j + V_l,$$
$$4V_{VI} = V_{IV} + V_g + V_i + V_V.$$

Перепишем эти уравнения так, чтобы потенциалы в граничных узлах с индексами от *a* до *n* оказались в правой части, и сохраним порядок, в каком они появились:

$$4V_I - V_{II} - V_{III} + 0V_{IV} + 0V_V + 0V_{VI} = V_b + V_n,$$
$$-V_I + 4V_{II} + 0V_{III} - V_{IV} + 0V_V + 0V_{VI} = V_c + V_e,$$
$$-V_I + 0V_{II} + 4V_{III} - V_{IV} - V_V + 0V_{VI} = V_m,$$
$$0V_I - V_{II} - V_{III} + 4V_{IV} + 0V_V - V_{VI} = V_f,$$
$$0V_I + 0V_{II} - V_{III} + 0V_{IV} + 4V_V - V_{VI} = V_j + V_l,$$
$$0V_I + 0V_{II} + 0V_{III} - V_{IV} - V_V + 4V_{VI} = V_g + V_i.$$

В матричной форме они приобретают следующий вид:

$$\begin{bmatrix} -4 & 1 & 1 & 0 & 0 & 0 \\ 1 & -4 & 0 & 1 & 0 & 0 \\ 1 & 0 & -4 & 1 & 1 & 0 \\ 0 & 1 & 1 & -4 & 0 & 1 \\ 0 & 0 & 1 & 0 & -4 & 1 \\ 0 & 0 & 0 & 1 & 1 & -4 \end{bmatrix} \begin{bmatrix} V_I \\ V_{II} \\ V_{III} \\ V_{IV} \\ V_V \\ V_{VI} \end{bmatrix} = -1 \cdot \begin{bmatrix} V_b + V_n \\ V_c + V_e \\ V_m \\ V_f \\ V_j + V_l \\ V_g + V_i \end{bmatrix}$$ или в матричном виде **Ax=b**.

Матрица **A**, имеет порядок *N* (число внутренних узлов). Отметим также, что матрица симметрична. Это обусловлено однородностью диэлектрического заполнения. Много элементов матрицы нулевые, порядка 40%. Подобная особенность характерна, как правило, для большинства систем высокого порядка из конечно-разностных уравнений. Соответствующий программный код приведен ниже.

```
clc;
clear;
A=[-4 1 1 0 0 0; 1 -4 0 1 0 0; 1 0 -4 1 1 0; 0 1 1 -4 0 1;
0 0 1 0 -4 1; 0 0 0 1 1 -4];
Vb=0;
Vn=15;
Vc=0;
```

```
Ve=5;
Vm=15;
Vf=5;
Vj=10;
Vl=15;
Vg=5;
Vi=10;
V=[Vb+Vn;Vc+Ve;Vm;Vf; Vj+Vl;Vg+Vi]
X=A\(-V)
```

Задание. Разобрать работу представленного программного кода и разработать программу при разбиении пространства на 36 подобластей.

1.3.4 Прямоугольная область: повышение точности

В ходе работы необходимо разработать программы для решения аналогичных задач, рассмотренных в трех предыдущих работах, при использовании девятиточечного шаблона. Также требуется оценить точность вычислений в сравнении с использованием пятиточечного шаблона. В конце сформировать выводы о проделанной работе.

1.3.5 Погонная емкость линии передачи

В ходе занятия требуется разработать программу для вычисления волнового сопротивления линии передачи (рисунок 1.10) без учета симметрии. Размеры структуры выбираются по согласованию с преподавателем.

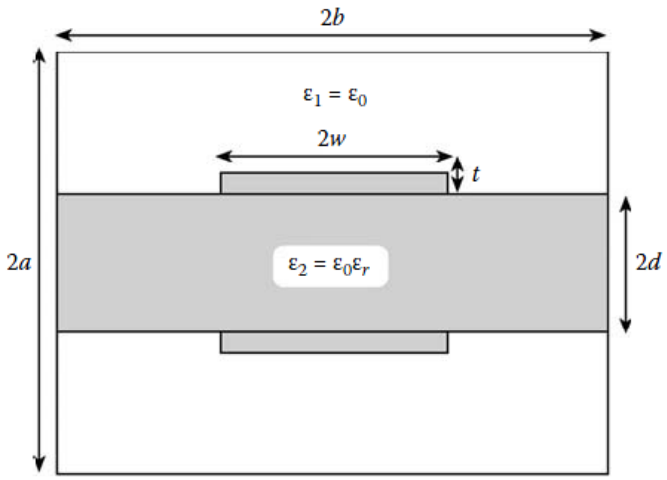


Рисунок 1.10 – Экранированная двухполосковая линия передачи (*а*) и её вид при использовании полной симметрии (*б*)

1.3.6 Напряженность электрического поля

Для линии передачи из предыдущего задания разработать программный код для вычисления значения компонент вектора напряженности электрического поля и их суммарных значений, а также их отображения.

После вычисления значений компонент вектора напряженности можно воспользоваться инструментарием Octave для наглядного представления полученных результатов figure и quiver(x, y, Ex, Ey).

1.4 Метод моментов

1.4.1 Кусочно-постоянные базисные функции

Требуется разработать программу для построения 4 кусочно-постоянных базисных функции (КПБФ), каждая из которых определяется следующим образом

$$f_m(x) = \begin{cases} 1, & x_m \le x \le x_{m+1} \\ 0, & \text{иначе} \end{cases}.$$

Для их одновременного отображения воспользоваться функциями subplot(m,n,p) и plot(). Первая задает массив для отображения графиков, состоящий из *m* строк и *n* столбцов. Последний индекс *p* предназначен для выбора ячейки данного массива, в которой будет отображен график с помощью последующей команды plot(). Для ознакомления с особенностями использования функции subplot() можно воспользоваться следующим простым примером:

```
>> subplot(2,2,1)
>> plot(1:5,1:5)
>> subplot(2,2,2)
>> plot(1:4,1:4)
>> subplot(2,2,3)
>> plot(2:5,1:4)
>> subplot(2,2,4)
>> plot(1:100,1:100)
```

Используя произвольно заданные весовые коэффициенты продемонстрировать правильность программной реализации (на одном графике). Далее модифицировать программный код для возможности задания

произвольного количества базисных функций.

1.4.2 Тонкая проволока

С учетом теоретических сведений, представленных на лекционных занятиях, а также с использованием программы, разработанной на предыдущем занятии, требуется разработать программу для вычисления линейной плотности заряда по поверхности тонкой проводящей проволоки длиной L и диаметром $2a$ ($2a << L$), ориентированной вдоль оси x (рисунок 1.11).

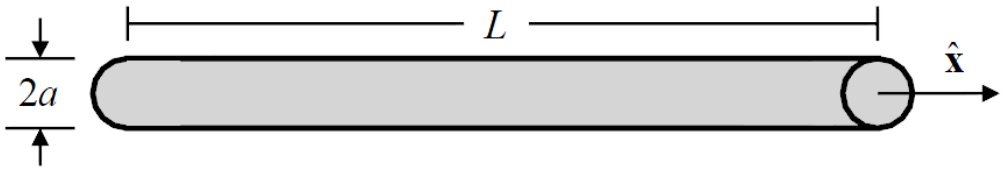


Рисунок 1.11 – Сегментация тонкой проволоки

1.4.3 Погонная ёмкость коаксиальной линии передачи

В системе TALGAT программно реализовать вычисление погонной емкости коаксиального кабеля (рисунок 1.12) при $a = b = 1$ см, $c = d = 2$ см. Относительная диэлектрическая проницаемость внутри линии равна 1. Выполнить оценку влияние уменьшение шага сегментации на точность результатов.

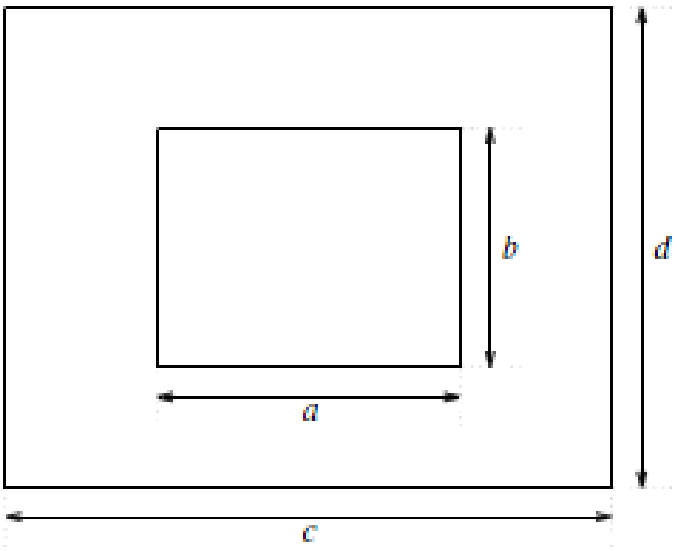


Рисунок 1.12 – Поперечное сечение коаксиального кабеля

1.4.4 Матрица погонного сопротивления линий передачи

Анализ линий передачи в квазистатическом приближении с помощью телеграфных уравнений основан на значениях матриц удельной емкости (**C**) и проводимости (**G**), удельной индуктивности (**L**) и сопротивления (**R**). В отличие от матриц емкости и проводимости, матрицы индуктивности и сопротивления проявляют значительную зависимость от частоты. Эта сильная зависимость от частоты вызвана скин-эффектом, из-за которого токи высокой частоты протекают преимущественно в поверхностных слоях проводников. Для вычисления матрицы **R** на высоких частотах в системе TALGAT реализован следующий алгоритм:

1. Ввод параметров материала проводника: ρ, μ.
2. Ввод частоты.
3. Ввод исходных геометрических параметров структуры.
4. Вычисление значения приращения границ проводника ∂n (по умолчанию используется значение 0,1 от минимального параметра структуры).
5. Вычисление поверхностного сопротивления проводника по формуле
$$R_s = \sqrt{\pi f \mu \rho}\,. \tag{1.7}$$
6. Построение геометрической модели поперечного сечения структуры при исходных параметрах.
7. Вычисление исходной матрицы погонных коэффициентов электромагнитной индукции **L1**.
8. Расширение всех границ опорного проводника на ∂n.
9. Вычисление матрицы индуктивностей **L2** для измененной структуры.
10. Вычисление $\Delta\mathbf{L}_{j,k} = \mathbf{L1}_{j,k} - \mathbf{L2}_{j,k}$.
11. Вычисление недиагональных элементов матрицы $\mathbf{R}_{j,k}$ по формуле:
$$\mathbf{R}_{j,k}\Big|_{j\neq k} = \frac{R_S}{\mu_0}\left(\frac{-\Delta\mathbf{L}_{j,k}}{\partial n}\right), \text{ Ом/м}. \tag{1.8}$$
12. Вычисление диагональных элементов матрицы $\mathbf{R}_{j,j}$, при j=1
13. Расширение всех границ j-го проводника.
14. Вычисление $\mathbf{L2}_{jj}$.

15. Вычисление $\Delta\mathbf{L}_{jj} = \mathbf{L1}_{jj} - \mathbf{L2}_{jj}$.
16. Вычисление диагональных элементов матрицы $\mathbf{R}_{jj}$:

$$\mathbf{R}_{jj} = \frac{R_s}{\mu_0}\left(\frac{-\Delta\mathbf{L}_{jj}}{\partial n}\right), \text{ Ом/м.} \quad (1.9)$$

17. Восстановление исходных границ *j*-го проводника.
18. $j=j+1$.
19. Повторение пунктов 13–17 для каждого диагонального элемента, поочередно расширяя поверхность вычисляемого проводника.

Таким образом, для вычисления матрицы **R** необходимо сначала вычислить матрицу СЛАУ с помощью команды SET "smn" SMN_L conf_ig. Затем вычислить матрицу **R**, с помощью команды: SET "Rm" CALCULATE_R smn conf_ig *f*. Здесь переменная *f* определяет значение частоты, на которой необходимо выполнить расчет. Глобальная переменная "delta_n_for_r" (строка 4 алгоритма, ∂n) определяется значением приращения границ проводника. (По умолчанию используется значение 0,1 относительно размера минимального параметра анализируемой структуры.) Далее приведен пример использования описанного функционала:

```
SET "delta_n_for_r" 1E-6
SET "smn" SMN_L conf_ig
SET "Rm" CALCULATE_R smn conf_ig 1.E10
ECHO TO_STRING Rm
```

В ходе работы необходимо вычислить матрицу погонных сопротивлений модального фильтра, поперечное сечение которого, приведено на рисунке 1.13. Использовать следующие неизменяемые значения параметров: t = 85 мкм, d = 1000 мкм, h = 400 мкм, ε_r = 5. Вычисления выполнить на частоте 1 ГГц (∂n – использовать по умолчанию) и определить оптимальную структуру (наименьшие значения матрицы сопротивлений) при изменении значения:

1. параметра s в диапазоне от 100, 120, …, 200 мкм при w = 500 мкм;
2. параметра w в диапазоне от 100, 200, …, 500 мкм при s = 200 мкм.

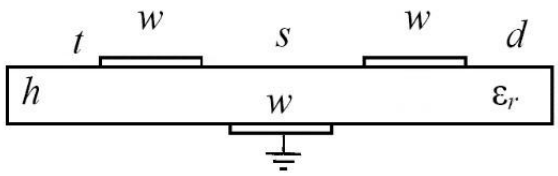


Рисунок 1.13 – Поперечное сечение модального фильтра №1

Далее приведен код программы для создания и отображения поперечного сечения модального фильтра.

```
INCLUDE "UTIL"
INCLUDE "MATRIX"
INCLUDE "MOM2D"
INCLUDE "INFIX"
SET_INFINITE_GROUND 0
SET "s" 0.0004
SET "w" 0.0003
SET "h" 290.0e-6
SET "t" 0.000105
SET "T_a" 0.
SET "Td1" 0.025
SET "ER1" 5.
SET "Vozd" 1.
SET "f0" 1.0e9
SET "Dtau_poln" 0.0000000025
SET_VARIABLE "d" 0.0003
SET "X1" DIV s 2.
SET "X2" PLUS X1 w
SET "X3" PLUS X2 d
SET "X4" MINUS 0. X1
SET "X5" MINUS 0. X2
SET "X6" MINUS 0. X3
SET "X7" DIV w 2.
SET "X8" MINUS 0. X7
SET "Y1" PLUS h t
SET "Y2" MINUS 0. t
SET_AUTO_SEGMENT_LENGTH 0.00001
CONDUCTOR
SET_ER_PLUS ER1
SET_TAN_DELTA_PLUS Td1
LINE X5 h X4 h
SET_ER_PLUS Vozd
SET_TAN_DELTA_PLUS T_a
LINETO X4 Y1
```

LINETO X5 Y1
LINETO X5 h
CONDUCTOR
SET_ER_PLUS ER1
SET_TAN_DELTA_PLUS Td1
LINE X1 h X2 h
SET_ER_PLUS Vozd
SET_TAN_DELTA_PLUS T_a
LINETO X2 Y1
LINETO X1 Y1
LINETO X1 h
CONDUCTOR_GROUNDED
SET_ER_PLUS Vozd
SET_TAN_DELTA_PLUS T_a
LINE X8 Y2 X7 Y2
LINETO X7 0.
SET_ER_PLUS ER1
SET_TAN_DELTA_PLUS Td1
LINETO X8 0.
SET_ER_PLUS Vozd
SET_TAN_DELTA_PLUS T_a
LINETO X8 Y2
DIELECTRIC
SET_ER_PLUS Vozd
SET_TAN_DELTA_PLUS T_a
SET_ER_MINUS ER1
SET_TAN_DELTA_MINUS Td1
LINE X6 0. X8 0.
LINE X7 0. X3 0.
LINETO X3 h
LINETO X2 h
LINE X1 h X4 h
LINE X5 h X6 h
LINETO X6 0.
SET "conf_ig" GET_CONFIGURATION_2D
DRAW_CONFIGURATION2D conf_ig

Далее необходимо вычислить матрицу погонных сопротивлений модального фильтра №2, поперечное сечение которого приведено на рисунке 1.14. Неизменяемые параметры: w = 1000 мкм, t = 35 мкм, s_2 = 685 мкм, d = 4000 мкм, ε_r = 4,5. Вычисления выполнить на частоте 1 ГГц (∂n – использовать по умолчанию) и определить оптимальную структуру

(наименьшие значения матрицы сопротивлений) при изменении значения:

1. параметра s_1 в диапазоне от 100, 150, …, 300 мкм при h = 500 мкм;
2. параметра h в диапазоне от 100, 200, …, 900 мкм при s_1 = 200 мкм.

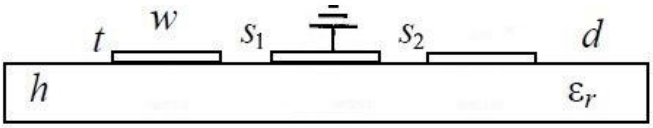


Рисунок 1.14 – Поперечное сечение модального фильтра №2

Далее приведен код программы для модального фильтра №2.

```
INCLUDE "MOM2D"
INCLUDE "UTIL"
INCLUDE "MATRIX"
INCLUDE "INFIX"
SET_AUTO_SEGMENT_LENGTH 25.0e-6
SET_INFINITE_GROUND 0
SET "w1" 1000.0e-6
SET "w2" 1000.0e-6
SET "w3" 1000.0e-6
SET "t" 35.e-6
SET "f0" 1.e6
SET "s1" 200.0e-6
SET "s2" 685.0e-6
SET "h" 500.e-6
SET "d" MUL 3. w1
SET "er1" 1.
SET "er2" 4.5
SET "dlina" 600.0e-3
SET "TdC" 0.017
SET "TdAir" 0.0
CONDUCTOR
SET_ER_PLUS er1
SET_TAN_DELTA_PLUS TdAir
LINE PLUS d w1 h PLUS d w1 PLUS h t
LINETO d PLUS h t
LINETO d h
SET_ER_PLUS er2
SET_TAN_DELTA_PLUS TdC
LINE d h PLUS d w1 h
CONDUCTOR_GROUNDED
SET_ER_PLUS er1
SET_TAN_DELTA_PLUS TdAir
```

LINE PLUS PLUS PLUS d w1 s1 w2 h PLUS PLUS PLUS d w1 s1 w2 PLUS h t
LINETO PLUS PLUS d w1 s1 PLUS h t
LINETO PLUS PLUS d w1 s1 h
SET_ER_PLUS er2
SET_TAN_DELTA_PLUS TdC
LINE PLUS PLUS d w1 s1 h PLUS PLUS PLUS d w1 s1 w2 h
CONDUCTOR
SET_ER_PLUS er1
SET_TAN_DELTA_PLUS TdAir
LINE PLUS PLUS PLUS PLUS PLUS d w1 s1 w2 s2 w3 h PLUS PLUS PLUS PLUS PLUS d w1 s1 w2 s2 w3 PLUS h t
LINETO PLUS PLUS PLUS PLUS d w1 s1 w2 s2 PLUS h t
LINETO PLUS PLUS PLUS PLUS d w1 s1 w2 s2 h
SET_ER_PLUS er2
SET_TAN_DELTA_PLUS TdC
LINE PLUS PLUS PLUS PLUS d w1 s1 w2 s2 h PLUS PLUS PLUS PLUS PLUS d w1 s1 w2 s2 w3 h
DIELECTRIC
SET_ER_PLUS er1
SET_TAN_DELTA_PLUS TdAir
SET_ER_MINUS er2
SET_TAN_DELTA_MINUS TdC
LINE d h 0. h
LINETO 0. 0.
LINE PLUS PLUS PLUS PLUS PLUS PLUS d w1 s1 w2 s2 w3 d 0. PLUS PLUS PLUS PLUS PLUS PLUS d w1 s1 w2 s2 w3 d h
LINETO PLUS PLUS PLUS PLUS PLUS d w1 s1 w2 s2 w3 h
LINE PLUS PLUS PLUS PLUS d w1 s1 w2 s2 h PLUS PLUS PLUS d w1 s1 w2 h
LINE PLUS PLUS d w1 s1 h PLUS d w1 h
SET "conf_ig_1" GET_CONFIGURATION_2D
DRAW_CONFIGURATION conf_ig_1

1.4.5 Адаптивная перекрестная аппроксимация

Метод моментов – эффективный численный метод решения интегральных уравнений, возникающих при моделировании проводящих объектов произвольной формы. Однако матрица СЛАУ (**ZI** = **V**), к решению которой сводится задача, является плотной. При этом хранение, заполнение и операции с матрицей **Z** имеют сложность $O(N^2)$, а её решение методом Гаусса (часто применяемым для решения СЛАУ с плотной матрицей) – $O(N^3)$, где N – количество неизвестных. Поэтому актуален поиск путей экономии

вычислительных ресурсов. Одним из таких путей является адаптивная перекрестная аппроксимация (ACA). Цель работы – исследование использования ACA при анализе полосковых структур методом моментов.

Суть ACA заключается в представлении некой матрицы **S** в виде произведения матриц **U** и **V** меньшего ранга. Преимущества данного алгоритма заключаются в его алгебраическом характере. За счет методов линейной алгебры, таких как QR-разложение, сингулярное разложение, LU-разложение и др. достигается ускорение вычислений. Благодаря тому, что алгоритм имеет алгебраический характер, он может быть модульным и легко интегрироваться в различные реализации метода моментов. Однако матрица, получаемая с его помощью, обладает сингулярностью на резонансных частотах, поэтому применить ACA к исходной матрице **S** не представляется возможным. В то же время, учитывая специфику функции Грина, данная матрица состоит из блоков, соответствующих взаимодействию хорошо сепарабельных базисных функций, благодаря чему они могут быть представлены в виде иерархических матриц. Таким образом, в общем случае, алгоритм ACA представляет собой многоуровневую систему сжатия матриц, которое обеспечивает ускорение вычислений. Стоит отметить, что алгоритм аппроксимирует исходную матрицу, требуя лишь частичную информацию о ней.

Пусть $\mathbf{S} = \mathbf{R} + \tilde{\mathbf{S}}$, где $\mathbf{R}$ – матрица ошибки аппроксимации, а $\tilde{\mathbf{S}}$ – матрица малого ранга (ранг $\tilde{\mathbf{S}} \le r$, $r << N$). При использовании ACA ищется матрица аппроксимации $\tilde{\mathbf{S}}$ плотной матрицы **S** (размера $N \times M$), в виде произведения двух матриц **U** и **V**. Т.е., $\tilde{\mathbf{S}} = \mathbf{UV} = \sum_{i=1}^{r} \mathbf{u}_i \mathbf{v}_i$, где **U** размера $M \times r$, а **V** размера $r \times N$, с точностью удовлетворяющей условию минимизации матрицы ошибки $\|\mathbf{R}\|_F = \|\mathbf{S} - \tilde{\mathbf{S}}\|_F \le \varepsilon \|\mathbf{S}\|_F$ (ε – требуемая точность).

Для ясности дальнейшего изложения введем обозначения (с использованием синтаксиса Octave). Пусть имеем матрицы $\mathbf{I} = [I_1 \ldots I_r]$ и $\mathbf{J} = [J_1 \ldots J_r]$, содержащие выбранные строчные и столбцовые индексы матрицы **S**. Для обозначения I_k строки, например, матрицы **R** используется запись $\mathbf{R}(I_k, :)$, а для

J_k столбца запись $\mathbf{R}(:, J_k)$. Тогда алгоритм ACA можно представить в виде:

1. Задать ε. Положить $\tilde{\mathbf{S}} = \mathbf{0}$ и $I_1 = 0$.
2. $\mathbf{R}(I_1, :) = \mathbf{S}(I_1, :)$
3. Найти индекс J_1, удовлетворяющий: $|\mathbf{R}(I_1, J_1)| = \max_j(|\mathbf{R}(I_1, j)|)$
4. $\mathbf{v}_1 = \mathbf{R}(I_1, :) / \mathbf{R}(I_1, J_1)$
5. $\mathbf{R}(:, J_1) = \mathbf{S}(:, J_1)$
6. $\mathbf{u}_1 = \mathbf{R}(:, J_1)$
7. $\| \tilde{\mathbf{S}}^{(1)} \|_F^2 = \| \tilde{\mathbf{S}}^{(0)} \|_F^2 + \| \mathbf{u}_1 \|_F^2 \| \mathbf{v}_1 \|_F^2$
8. Найти индекс I_2, удовлетворяющий: $|\mathbf{R}(I_2, J_1)| = \max_i(|\mathbf{R}(i, J_1)|)$, $i \neq I_1$
9. **Для** $k = 2, 3, \ldots$, вычислить
10. $\mathbf{R}(I_k,:) = \mathbf{S}(I_k,:) - \sum_{l=1}^{k-1} (\mathbf{u}_l)_{I_k} \mathbf{v}_l$
11. Найти J_k, удовлетворяющий: $|\mathbf{R}(I_k, J_k)| = \max_j(|\mathbf{R}(I_k, j)|)$, $j \neq J_1, \ldots, J_{k-1}$
12. **Если**
13. $\mathbf{R}(I_k, J_k) = 0$, то **конец**, аппроксимация не получена
14. **Иначе**
15. $\mathbf{v}_k = \mathbf{R}(I_k, :) / \mathbf{R}(I_k, J_k)$
16. $\mathbf{R}(:, J_k) = \mathbf{S}(:, J_k) - \sum_{l=1}^{k-1} (\mathbf{v}_l)_{J_k} \mathbf{u}_l$
17. $\mathbf{u}_k = \mathbf{R}(:, J_k)$
18. $\| \tilde{\mathbf{S}}^{(k)} \|_F^2 = \| \tilde{\mathbf{S}}^{(k-1)} \|_F^2 + 2 \sum_{j=1}^{k-1} | \mathbf{u}_j^T \mathbf{u}_k | | \mathbf{v}_j^T \mathbf{v}_k | + \| \mathbf{u}_k \|_F^2 \| \mathbf{v}_k \|_F^2$
19. **Если**
20. $\|\mathbf{u}_k\|_2 \|\mathbf{v}_k\|_2 \leq \varepsilon \| \tilde{\mathbf{S}}^{(k)} \|_2$, то **конец** итерационного процесса, $r = k$
21. **Иначе**
22. Найти индекс I_{k+1}, удовлетворяющий равенству $|\mathbf{R}(I_{k+1}, J_k)| = \max_i(|\mathbf{R}(i, J_k)|)$, $i \neq I_1, \ldots, I_k$
23. **Увеличить** k

Из алгоритма видно, что для построения аппроксимации не нужно заранее знать всех элементов исходной матрицы **S**. Для выполнения вычислений с помощью данного алгоритма требуется не более r итераций, сложность каждой из которых пропорциональна $O(r \times (M + N))$. Таким образом, общие вычислительные затраты алгоритма пропорциональны $O(r^2 \times (M + N))$. Далее приведен программный код Octave, предназначенный для считывания матрицы и её (или её блоков) разложения.

```
function ACA_main
clc;
clear;
```

```
A=dlmread('matrix_forACA_.txt');
b=A;
[n,m]=size(b);
eps_tol=1.e-3;
 [U,V] = ACA(b,eps_tol,n,m);
size(U)
size(V)
norm(b-U*V)
function [U,V] = ACA(A,ACA_tol,M,N)
  J = zeros(N,1);
  I = zeros(M,1);
  i = (2:M);
  j = (1:N);
  I(1) = 1;
  Rik=A(I(1),:);
  col = find( abs(Rik(j)) == max(abs(Rik(j))) );
  J(1) = j(col(1));
  j(j==J(1))=[];
  V = Rik/Rik(J(1));
  Rjk=A(:,J(1));
  U = Rjk;
  normZ = norm(U)^2 * norm(V)^2;
  row = find( abs(Rjk(i)) == max(abs(Rjk(i))) );
  I(2) = i(row(1));
  i(i==I(2))=[];
  for k=2:min(M,N)
     Rik=A(I(k),:)- U(I(k),:)*V;
     col = find(abs(Rik(j)) == max(abs(Rik(j))) );
     J(k) = j(col(1));
     j(j==J(k))=[];
     if(Rik(J(k)) == 0)
        break;
     end
     Vk = Rik/Rik(J(k));
     Rjk=A(:,J(k)) - U*V(:,J(k));
     Uk = Rjk;
     normZ = normZ + 2*sum(real((U'*Uk).*(Vk*V').')) + norm(Uk)^2*norm(Vk)^2;
     U = [U Uk]; V = [V; Vk];
     if norm(Uk)*norm(Vk) <= ACA_tol*sqrt(normZ)
        break
     end
     if k==min(M,N)
        break;
     end
```

```
    row = find( abs(Rjk(i)) == max(abs(Rjk(i))) );
    I(k+1) = i(row(1));
    i(i==I(k+1))=[];
  end
```

В ходе работы необходимо изучить программный код и исследовать работу ACA на примере тестовой матрицы. Для генерации, которой необходимо воспользоваться системой TALGAT (результат работы – файл *matrix_forACA_.txt*, содержащий матрицу размера 400 × 400):

```
INCLUDE "MOM2D"
INCLUDE "MATRIX"
INCLUDE "UTIL"
SET "w" 100.e-6 //ширина проводника
SET "s" MUL 1. w //расстояние между проводниками
SET "t" 25.e-6 //толщина проводников
SET "d" MUL 3. w
SET "hC" 30.e-6
SET "ErAir" 1.0
SET "TdV" 0.0
SET "TdAir" 0.0
SET "segm" DIV t 20.
SET_AUTO_SEGMENT_LENGTH segm
SET_INFINITE_GROUND 1
CONDUCTOR
SET_ER_PLUS ErAir
LINE 0. hC 0. PLUS hC t
LINETO w PLUS hC t
LINETO w hC
LINETO 0. hC
CONDUCTOR
SET_ER_PLUS ErAir
LINE PLUS w s hC PLUS w s PLUS hC t
LINETO PLUS PLUS w w s PLUS hC t
LINETO PLUS PLUS w w s hC
LINETO PLUS w s hC
SET "conf_ig" GET_CONFIGURATION_2D
DRAW_CONFIGURATION conf_ig
SET "smn" SMN_C conf_ig
SET "mCG" CALCULATE_C smn conf_ig f
ECHO mCG
DRAW_CONFIGURATION conf_ig
OPEN_FOR_WRITE matrix_forACA_.txt
```

WRITE TO_STRING smn
CLOSE

Аппроксимацию применить:

- ко всей матрице (b=A);
- к блокам матрицы b=A(1:200,201:400) и b=A(201:400,1:200);
- к блокам матрицы b=A(1:100,101:200) и b=A(101:200,1:100);
- к блокам матрицы b=A(201:300,301:400) и b=A(301:400,201:300).

Полученные результаты проанализировать и сформулировать рекомендации по использованию АСА.

1.5 Метод конечных элементов

1.5.1 Вычисление значений потенциала

Пусть дана двухэлементная сетка, представленная на рисунке 1.15*а*. Требуется найти значения потенциала внутри сетки, используя метод конечных элементов.

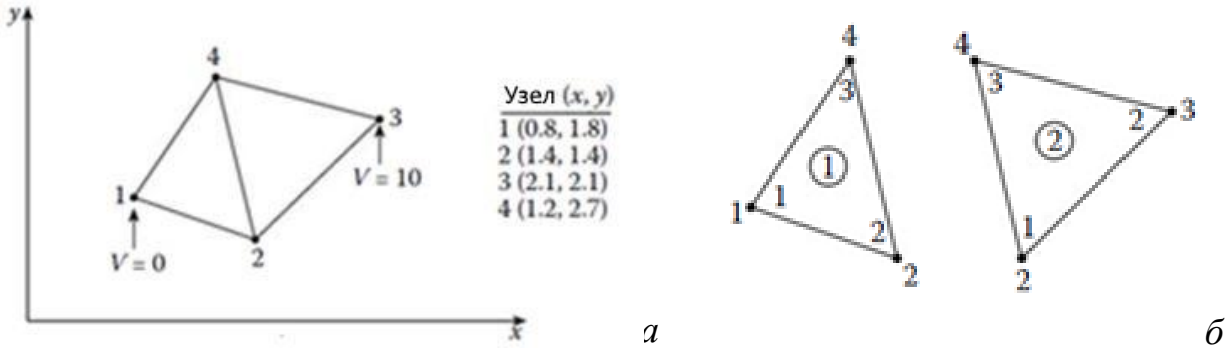


Рисунок 1.15 – Конечно-элементная сетка (*а*),
локальная и глобальная нумерация (*б*)

Вычисление матрицы коэффициентов элементов может быть упрощено, если положить

$$P_1 = (y_2 - y_3),\ P_2 = (y_3 - y_1),\ P_3 = (y_1 - y_2),$$
$$Q_1 = (x_3 - x_2),\ Q_2 = (x_1 - x_3),\ Q_3 = (x_2 - x_1), \qquad (1.10)$$

где P_i и Q_i (i = 1, 2, 3 – номера локальных узлов). Тогда каждый элемент матрицы может быть вычислен с помощью выражения

$$c_{ij}^{(e)} = \frac{1}{4S}\left(P_i P_j + Q_i Q_j\right), \qquad (1.11)$$

где $S = 0{,}5(P_2Q_3 - P_3Q_2)$. Так, для элемента 1, содержащего глобальные узлы 1-2-4, соответствующие локальным узлам 1-2-3 (рисунок 1.15*б*), получим

$$P_1 = (1{,}4 - 2{,}7) = -1{,}3;\ P_2 = (2{,}7 - 1{,}8) = 0{,}9;\ P_3 = (1{,}8 - 1{,}4) = 0{,}4;$$

$$Q_1 = (1{,}2 - 1{,}4) = -0{,}2;\ Q_2 = (0{,}8 - 1{,}2) = -0{,}4;\ Q_3 = (1{,}4 - 0{,}8) = 0{,}6;$$

$$S = 0{,}5(0{,}54 + 0{,}16) = 0{,}35.$$

Подставив полученные значения в (1.11), получим

$$\mathbf{C}^{(1)} = \begin{bmatrix} 1{,}2357 & -0{,}7786 & -0{,}4571 \\ -0{,}7786 & 0{,}6929 & 0{,}0857 \\ -0{,}4571 & 0{,}0857 & 0{,}3714 \end{bmatrix}. \qquad (1.12)$$

Аналогично, для элемента 2, содержащего глобальные узлы 2-3-4, соответствующие локальным 1-2-3 (рисунок 1.15*б*), получим

$$P_1 = -0{,}6;\ P_2 = 1{,}3;\ P_3 = -0{,}7;$$

$$Q_1 = -0{,}9;\ Q_2 = 0{,}2;\ Q_3 = 0{,}7;$$

$$S = 0{,}5(0{,}91 + 0{,}14) = 0{,}525.$$

Нужно помнить, что данные значения получены из расчета однородного диэлектрического заполнения. При неоднородном заполнении необходимо внести корректировки путем их умножения на значение диэлектрической проницаемости, соответствующее рассматриваемому конечно элементу. Подставив данные значения в (1.11), получим

$$\mathbf{C}^{(2)} = \begin{bmatrix} 0{,}5571 & -0{,}4571 & -0{,}1 \\ -0{,}4571 & 0{,}8238 & -0{,}3667 \\ -0{,}1 & -0{,}3667 & 0{,}4667 \end{bmatrix}. \qquad (1.13)$$

Важным фактом является то, что нужно придерживаться локальной нумерации при вычислении элементов матрицы отдельного элемента и глобальной нумерации – глобальной матрицы. В результате элементы глобальной матрицы вычисляются следующим образом:

$$\mathbf{C}_{22} = \mathbf{C}_{22}^{(1)} + \mathbf{C}_{11}^{(2)} = 0{,}6929 + 0{,}5571 = 1{,}25;$$
$$\mathbf{C}_{24} = \mathbf{C}_{23}^{(1)} + \mathbf{C}_{13}^{(2)} = 0{,}0857 - 0{,}1 = -0{,}0143;$$
$$\mathbf{C}_{44} = \mathbf{C}_{33}^{(1)} + \mathbf{C}_{33}^{(2)} = 0{,}3714 + 0{,}4667 = 0{,}8381;$$
$$\mathbf{C}_{21} = \mathbf{C}_{21}^{(1)} = -0{,}7786;$$
$$\mathbf{C}_{23} = \mathbf{C}_{12}^{(2)} = -0{,}4571;$$
$$\mathbf{C}_{41} = \mathbf{C}_{31}^{(1)} = -0{,}4571$$
$$\mathbf{C}_{43} = \mathbf{C}_{32}^{(2)} = -0{,}3667.$$

В результате получим глобальную матрицу:

$$\mathbf{C} = \begin{bmatrix} c_{11}^{(1)} & c_{12}^{(1)} & 0 & c_{13}^{(1)} \\ c_{21}^{(1)} & c_{22}^{(1)} + c_{11}^{(2)} & c_{12}^{(2)} & c_{23}^{(1)} + c_{12}^{(2)} \\ 0 & c_{21}^{(2)} & c_{22}^{(2)} & c_{23}^{(2)} \\ c_{31}^{(1)} & c_{32}^{(1)} + c_{31}^{(2)} & c_{32}^{(2)} & c_{33}^{(1)} + c_{33}^{(2)} \end{bmatrix} =$$
$$= \begin{bmatrix} 1{,}2357 & -0{,}7786 & 0 & -0{,}4571 \\ -0{,}7786 & 1{,}25 & -0{,}4571 & -0{,}0143 \\ 0 & -0{,}4571 & 0{,}8238 & -0{,}3667 \\ -0{,}4571 & -0{,}0143 & -0{,}3667 & 0{,}8381 \end{bmatrix}. \qquad (1.14)$$

Видно, что $\sum_{i=1}^{4} c_{ij} = \sum_{j=1}^{4} c_{ij} = 0$, что говорит о правильности вычисления элементов матрицы.

В общем случае, для сетки, состоящей из n узлов в k узле, имеем

$$\Phi_k = -\frac{1}{c_{kk}} \sum_{i=1, i\neq k}^{n} \Phi_i c_{ki}, \qquad (1.15)$$

где узел k является свободным. Если между узлами k и i нет прямой связи, то $c_{ki} = 0$. Таким образом, только узлы, которые имеют прямую связь с узлом k, вносят вклад в значение потенциала (Φ_k) согласно (1.15). Уравнение (1.15) можно применить итерационно ко всем свободным узлам. Так, итерационный процесс начинается с задания значений потенциалов в фиксированных (граничных) узлах. Тогда применив уравнение (1.15) к свободным узлам 2 и 4, получим

$$\Phi_2 = -\frac{1}{c_{22}}\left(\Phi_1 c_{12} + \Phi_3 c_{32} + \Phi_4 c_{42}\right),$$

$$\Phi_4 = -\frac{1}{c_{44}}\left(\Phi_1 c_{14} + \Phi_2 c_{24} + \Phi_3 c_{34}\right),$$

или

$$\begin{aligned} \Phi_2 &= -\frac{1}{1{,}25}\left(-4{,}571 - 0{,}0143\Phi_4\right), \\ \Phi_4 &= -\frac{1}{0{,}8381}\left(-0{,}143\Phi_2 - 3{,}667\right). \end{aligned} \qquad (1.16)$$

Используя, например, нулевое начальное приближение и итерационно формулы (1.16), получим после первой итерации $\Phi_2 = 3{,}6568$, $\Phi_4 = 4{,}4378$, а после второй $\Phi_2 = 3{,}7075$, $\Phi_4 = 4{,}4386$.

Как только значения потенциалов в узлах известны, тогда может быть определен потенциал в любой точке сетки с помощью следующих уравнений:

$$\Phi_e = \sum_{i=1}^{3} \alpha_i(x, y)\Phi_{ei}$$

где

$$\alpha_1 = \frac{1}{2S}\left[\left(x_2 y_3 - x_3 y_2\right) + \left(y_2 - y_3\right)x + \left(x_3 - x_2\right)y\right],$$

$$\alpha_2 = \frac{1}{2S}\left[\left(x_3 y_1 - x_1 y_3\right) + \left(y_3 - y_1\right)x + \left(x_1 - x_3\right)y\right],$$

$$\alpha_3 = \frac{1}{2S}\left[\left(x_1 y_2 - x_2 y_1\right) + \left(y_1 - y_2\right)x + \left(x_2 - x_1\right)y\right],$$

а S – площадь элемента e, т.е.

$$S = \frac{1}{2}\left[\left(x_2 - x_1\right)\left(y_3 - y_1\right) - \left(x_3 - x_1\right)\left(y_2 - y_1\right)\right].$$

1.5.2 Вычисление значений потенциала: программная реализация

По результатам предыдущего занятия требуется разработать программу для вычисления значений потенциала на примере двухэлементной сетки, представленной на рисунке 1.15*а*. Далее приведен программный код,

снабженный сопутствующими комментариями и реализующий решение поставленной задачи.

```
% ************************************************
% Шаг 1 - Ввод данных: описание геометрии и граничных условий
% ************************************************
clear;
clc;
NI = 50; %число итераций
tol=1e-6;
NE=2; %число КЭ
ND=4; %число глобальных узлов
NP=2; %число КЭ, значение потенциала в которых известно
X = [0.8 1.4 2.1 1.2]; %x-координаты узлов
Y = [1.8 1.4 2.1 2.7]; %y-координаты узлов
NDP = [1 3]; %номера узлов, в которых значение потенциала известно
VAL = [0 10]; %известные значения потенциала в узлах
NL = [%номера узлов каждого КЭ
   1 2 4 ;
   2 3 4    ];
% ************************************************
% Шаг 2 - Вычисление матриц коэффициентов и ансамблирование
% ************************************************
C = zeros(ND,ND); % глобальная матрица C
for I = 1: NE %цикл по количеству КЭ
      % определение локальных координат XL, YL для I-го элемента
      XL = X(NL(I,:));
      YL = Y(NL(I,:));
      %формулы 1.7
      P = [YL(2) - YL(3) , YL(3) - YL(1) , YL(1) - YL(2)];
      Q = [XL(3) - XL(2) , XL(1) - XL(3) , XL(2) - XL(1)];

      AREA = 0.5*( P(2)*Q(3) - Q(2)*P(3));
      % Вычисление матрицы коэффициентов каждого I-го элемента
      % (в данном случае, обоих КЭ) согласно формуле 1.8
      CE = (P'*P+Q'*Q)/(4*AREA); % результат 1.9 и 1.10
      % Ансамблирование
      IR = NL(I,:);
      C(IR,IR) = C(IR,IR) + CE; %результат 1.11
end
% ************************************************
% Шаг 3 - итерационное вычисление значений потенциала в требуемых узлах
% ************************************************
LF = setdiff(1:ND, NDP); %определение номеров узлов, в которых значения
```

```
потенциала не известны
V = zeros(1,ND);%
V(NDP) = VAL; % задание потенциалов в предписанных узлах
NF = length(LF);%вычисление числа узлов, в которых значения не известны
for N = 1:NI %итерационный цикл
   for ii = 1:NF
      %реализация формулы 1.12
      NC=1:ND;
      NC(LF(ii))=[];
      V(LF(ii)) = -1/(C(LF(ii),LF(ii)))*V(NC)*C(NC,LF(ii));
   end
end
% ************************************************
% Шаг 4 – вывод результатов вычислений
% ************************************************
disp('Values of potential in nodes');
disp(V')
```

Данная программа итерационно вычисляет значения потенциала в требуемых узлах. Необходимо проработать программный код и модернизировать его для останова итераций при достижении точности вычислений 10^{-9}, определяемой относительной погрешностью вектора потенциалов.

1.6 Валидация результатов моделирования

Рассмотрим стандарт IEEE 1597.1, в котором определен процесс проверки (валидации) вычислительных методов, средств компьютерного моделирования и математических моделей, применяемых при решении электромагнитных задач. Стандарт применим к широкому кругу задач: ЭМС, эффективная поверхность рассеяния, целостность сигнала, антенны. Проверка данных решения достигается путем сравнения с наборами данных, полученными при измерениях, альтернативными программными кодами, аналитическими выражениями и др. Рекомендуемая практика применения этого стандарта описана в стандарте IEEE 1597.2 от 2010 г.

Процесс проверки (валидации) компьютерного моделирования и моделирований (симуляции), описанный в стандарте, во многом зависит от

того, что именно проверяется и какие данные доступны для подтверждения. Предпочтительной является валидация результатов неизвестного случая с помощью известных эталонных решений (external reference). С этой целью в стандарте описаны различные способы выбора таких решений, а также процедура сравнения полученных наборов данных. Если подходящее сравнение не может быть получено, то валидация возможна с использованием схем, основанных на проверке в том же программном продукте при других параметрах – внутренняя проверка (self-referenced). Хотя эти схемы и имеют меньший охват, но они, тем не менее, позволяют достичь высокого уровня точности. Блок-схема (оригинальная версия), показанная на рисунке 1.16, отражает последовательность действий процесса валидации.

Существует три разных уровня полной валидации модели. При выборе способа важно учитывать, какого уровня необходимо достичь.

1. Математический уровень: валидация вычислительных методов.

2. Уровень реализации: валидация конкретного программного кода.

3. Уровень модели: валидация конкретной модели.

Первый уровень определяет корректность выбранного вычислительного метода. Второй включает конкретную реализацию кода, а третий – конкретные детали модели. Для полной проверки модели все три уровня должны быть корректными.

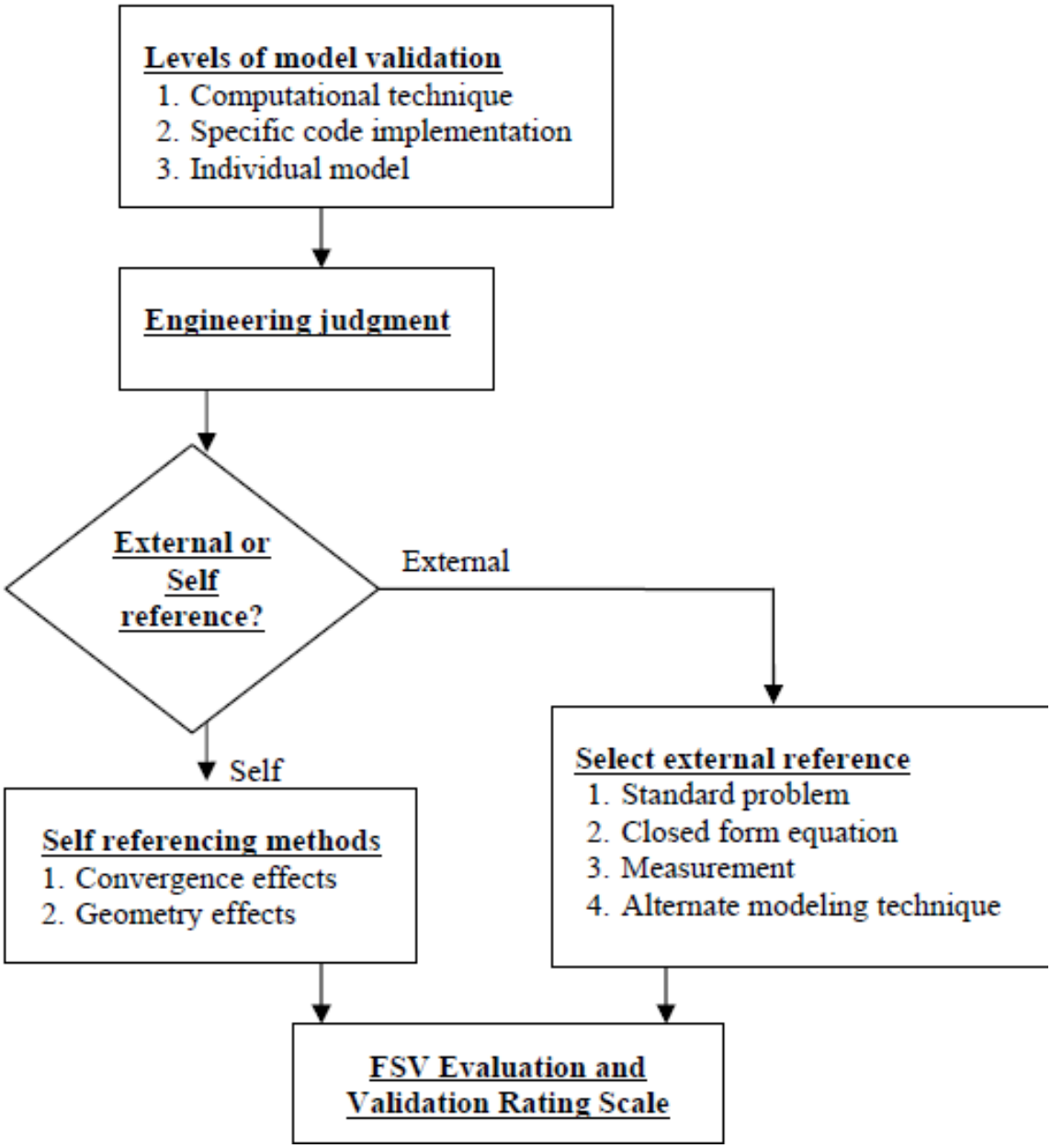


Рисунок 1.16 – Блок схема процесса валидации

Первым уровнем является валидация вычислительного метода. Обычно этот шаг не является необходимым при решении большей части задач электромагнитного моделирования, так как вычислительные методы подвергаются обширной экспериментальной проверке. В том случае, если была разработана новая методика, она должна пройти валидацию с целью определения ограничений, сильных сторон, а также точности метода. При валидации кода или нового вычислительного метода может потребоваться широкий диапазон моделей для полной проверки.

Второй уровень валидации состоит в обеспечении достоверности

программной реализации и получении корректных результатов для определенной модели. Рекомендуется проверять конкретные части кода на примере решения типовых задач, для решения которых они будут использоваться.

Третий уровень валидации предназначен для конкретной модели и является наиболее распространенной проблемой для инженеров. Практически во всех случаях инструменты компьютерного моделирования предоставят очень точный ответ на заданный вопрос. Однако нет никакой гарантии, что был задан правильный «вопрос», т.е. пользователь может непреднамеренно указать источник или какой-либо другой элемент модели, который не является значимым для фактической физической структуры.

Перед проведением детального анализа валидации необходимо убедиться, что выбор проверяемой модели является надёжным. Приведенный далее список не является исчерпывающим, т.к. другие проверки могут быть более подходящими для конкретной модели, однако, если какая-либо из проверок применима, временные затраты могут быть сокращены.

Некоторые из методов моделирования (MoM и PEEC) вычисляют токи по всей структуре. Поля, определяемые токами, которые навелись при воздействии на неё, обеспечивают существенное понимание валидности вычислительного результата. Просмотр токов на определенных частотах, особенно вблизи резонанса, позволяет наблюдать картины стоячих волн, поведение в областях разрывов и разрывы в металлических поверхностях. Токи не должны быстро меняться в соседних участках или сегментах и должны иметь близкое к нулю значение на концах проводов или пластин. Если эти требования не выполняются, то это указывает на наличие проблем при построении модели.

При использовании методов моделирования во временной области (FDTD, PEEC, TLM) значения полей, токов или напряжений находятся для всей вычислительной области для каждого временного шага. Как правило, конечным результатом является напряженность поля в определенном месте. Однако просмотр полученных значений во всей области вычислений может дать

существенное представление о валидности вычислительного результата.

При использовании методов моделирования во временной области (например, FDTD и TLM), необходимо убедиться в корректности заданных граничных условий во всех точках наблюдения, чтобы гарантировать, что поля не отражаются от границ вычислительной области. Наблюдение за конечными результатами поля в какой-то конкретной точке не является гарантией того, что все возможные эффекты были правильно смоделированы.

Методы моделирования во временной области, основанные на токе или напряжении (такие как PEEC и TLM), особенно эффективны при анализе печатных плат. Можно просмотреть анимацию токов или напряжений, чтобы гарантировать, что сигнал распространяется по требуемому пути и что любые резонансы затухают до определенного значения, и тем самым доказать, что моделирование завершено.

В некоторых случаях для проверки модели могут быть использованы известные величины. Например, значения диаграммы направленности диполя хорошо известны, и если модель похожа на диполь, то результаты моделирования диполя могут дать представление о валидности первичной модели.

Зачастую проверки работоспособности могут быть выполнены на основе математических «граничных условий». Примерами являются проверка симметрии полученного электромагнитного поля в симметричной задаче, проверка согласованности решения поля с уравнениями Максвелла и проверка согласованности полевых решений с производными уравнениями (например, сохранение заряда и энергии).

После определения того, какие результаты моделирования подлежат проверке, необходимо выбрать соответствующий подход. Там, где это возможно, будет проведено сравнение полученного решения с эталонным. Решения стандартных проблем доступны для валидации исходных моделей. В тех случаях, когда они недостаточны, эталонное решение может быть получено с помощью выражений в замкнутом виде, измерений или альтернативных

методов моделирования. В ситуациях, когда невозможно провести сравнение, необходимо убедиться в точности модели с помощью того же программного продукта при задании других параметров моделирования (внутренняя проверка). К ним относятся оценка сходимости и оценка геометрии задачи. Выбор подходящего подхода имеет решающее значение, поскольку он является основой валидации. Поэтому крайне важно использовать подход, дающий максимально возможную точность.

В последние годы разработано несколько вариантов валидаций стандартных задач, созданных для того, чтобы помочь инженерам. Решения задач известны и могут быть использованы в качестве эталона для проверки модели.

Выбор модели, геометрия которой достаточно проста и позволяет использовать выражения в замкнутом виде, должен выполняться с осторожностью. Например, точное моделирование отражений от идеально проводящей сферы может служить хорошим показателем, но не гарантирует, что моделирование отражений от сложного объекта, такого как самолет или танк, даст правильный результат. Кроме того, эти выражения не отражают того, как код будет оценивать эмиссии от печатной платы или вычислять эффективность экранирования. Выражения в замкнутом виде, как правило, являются специализированными и не отражают общую задачу, решение которой представляет интерес для пользователя. Поэтому рекомендации на основании этих выражений должны быть выбраны с осторожностью, чтобы обеспечить их соответствие задачам моделирования.

При использовании результатов измерений в качестве основы для валидации необходимо проявлять большую осторожность, чтобы минимизировать погрешность измерений. Например, измерения антенн и эффективной поверхности рассеяния могут быть выполнены с высокой точностью, в то время как при измерении электромагнитных помех могут быть получены погрешности, превышающие 6 дБ. Чтобы свести к минимуму количество ошибок валидации, связанных с использованием результатов

измерений, геометрические размеры вычислительной среды должны быть точно представлены в модели, общая точность измерений должна быть соблюдена, должно быть учтено влияние измерительного оборудования.

Другим популярным подходом к валидации результатов моделирования является моделирование одной и той же проблемы с использованием двух разных методов. Если физические свойства проблемы правильно смоделированы с помощью обоих методов, то результаты должны совпадать. Достижение одинаковых результатов более чем одним методом моделирования для одной и той же задачи может добавить уверенность в валидность результатов. Сравнение метода моделирования на основе решения дифференциальных уравнений (FDTD, FEM, TLM) и метода на основе решения интегральных уравнений (MoM, PEEC) предпочтительно из-за различной природы этих подходов.

По самой природе методов полноволнового моделирования часто при симуляции наблюдаются резонансы, объясняющиеся геометрией структуры. Эти резонансы важны для обоснованности результатов моделирования. Чаще всего моделирование практических задач сводится к раздельному моделированию небольших частей объекта из-за ограничений памяти и сложности модели. Эти небольшие модели также имеют резонансы, которые не имеют отношения к резонансам объекта в целом. Результаты, основанные на этих резонансах, часто вводят в заблуждение, поскольку резонанс не обусловлен изучаемым явлением, а скорее обусловлен размерами разделенной модели. Поэтому следует проявлять осторожность при оценке валидности модели несколькими методами.

Если модель не может быть проверена с помощью эталонного решения, то должна быть применена внутренняя проверка. Это менее надежный метод, из-за возможного сохранения ошибки в исходной модели, поэтому он должен использоваться только в тех случаях, когда сравнение с эталонным решением невозможно.

Вторичные модели создаются с помощью тех же подходов, что и

первичные, и используются в качестве основы для проверки первичных моделей. Требуется создать, по меньшей мере, два варианта вторичных моделей, однако не исключена необходимость создания дополнительных моделей с целью более точного выявления источников ошибок. Существуют два вида изменений, которые могут быть внесены в параметры первичной модели, чтобы создать валидные вторичные модели. Изменения не должны существенно влиять на результат и должны иметь известное воздействие на решение. Параметры оценки сходимости решения: сетка, сегмент или размер ячейки; вычислительная область; абсорбирующие граничные условия. Параметры оценки изменения геометрии: размер апертур; количество апертур; размещение компонентов на печатных платах. Результаты, полученные с помощью первичной и вторичных моделей, сравниваются с использованием метода FSV.

Существует ряд параметров модели, которые должны быть определены до того, как будет выполнено фактическое моделирование. Размер сетки или ячейки часто устанавливается не более чем одна десятая длины волны, исходя из предположения о том, что токи или поля постоянны внутри них. Однако этот размер может быть недостаточно мал, чтобы правильно зафиксировать изменение тока или поля, если их амплитуда быстро изменяется на структуре. Изменение размера сетки или ячейки является хорошим способом обеспечения их правильного размера. Если размер сетки или ячейки правильный, окончательные результаты моделирования не изменяются. Однако, если результаты изменяются при изменении размера сетки или ячейки, то использован неправильный размер.

При использовании объемных методов (FDTD, FEM, TLM) может варьироваться размер области вычислений. Вариация области вычислений может обеспечить отсутствие ложных сигналов и эффекта усечении сетки возле поглощающих границ, которые влияют на физическую модель. Конечный результат не должен зависеть от размера области вычисления или расстояния между физической моделью и поглощающими границами.

Полезно использовать изменения в исходных моделях для оценки конкретных изменений в решении. Модели обычно содержат ряд параметров, влияющих на результат. Размер апертур, их количество и размещение компонентов на печатных платах могут влиять на конечный результат моделирования. Во многих случаях последствия изменения параметра можно предсказать из опыта, хотя количественное изменение может быть заранее неизвестно.

После выбора подхода для валидации, необходимо использовать метод для её оценки. Метод валидации выделением особенностей (FSV) используется для определения соответствия между эталонным решением и полученными результатами моделирования.

В прошлом для сравнения двух наборов данных использовались различные методы. Простое вычитание одного набора данных из второго показывает разницу между значениями, но ограничено в качестве индикатора согласования набора данных, в случае если существует небольшое смещение между наборами данных. Подобным образом была использована взаимная корреляция, однако её результаты трудно сопоставимы с результатами визуальной проверки наборов данных, выполненной человеком.

FSV – метод, сочетающий в себе сравнение на основе амплитуд с сравнением, основанным на характеристиках, чтобы дать общее представление о согласованности двух наборов данных. FSV был откалиброван чтобы соответствовать решениям эксперта, которые несколько субъективны, и ранжировать согласованность по уровням: «отлично», «очень хорошо», «хорошо», «плохо» и т.д. Метод FSV используется для количественной оценки сопоставления наборов данных для валидации. Результат применения метода состоит в нахождении глобальной меры различия (global difference measure, GDM). Дополнительные уровни детализации метода FSV могут быть применены, чтобы помочь в сравнении и понимании любого несоответствия между наборами данных.

Опытные профессионалы могут посмотреть на данные, представленные на

рисунке 1.17, и решить, что два набора имеют «хорошую» или «нормальную» согласованность в зависимости от их ожиданий. FSV позволяет объективно выполнить сравнение, удаляя элемент субъективности из процесса принятия решений.

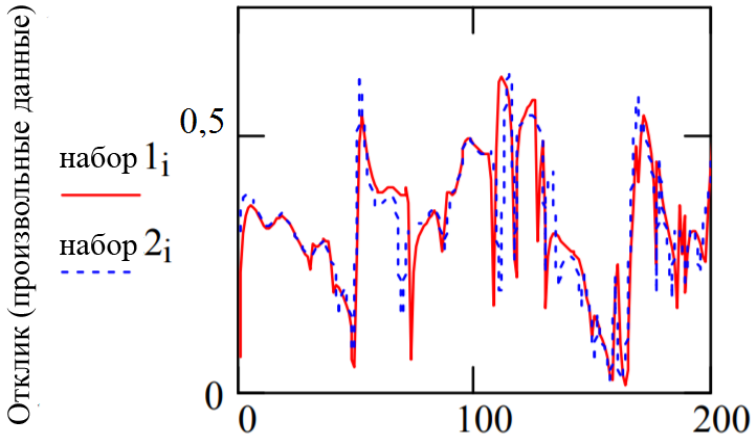


Рисунок 1.17 – Пример сравнения двух наборов данных

Основой метода FSV является декомпозиция результатов сравнения на две «составные» меры, а затем рекомбинация результатов, с целью получения глобальной степени соответствия. В методе задействована мера разности амплитуд (ADM), которая сравнивает амплитуды и характер изменения двух наборов данных, и мера разности особенностей (FDM), которая сравнивает быстро меняющиеся параметры (как функции независимых переменных). После этого методы ADM и FDM объединяются для формирования GDM. Величины ADM, FDM и GDM можно использовать в качестве инструментов для детального анализа или как единого общего измерения.

Шкала оценки валидации (VRS) устанавливает метод сравнительного анализа, позволяющий проводить «визуальные» сравнения между экспериментальными и моделируемыми данными (или любыми двумя наборами данных). Этот метод основан на общей количественной оценке подобия. Цель VRS двойственна: во-первых, они позволяет отдельным лицам или группам экспертов визуально оценивать сравнение с помощью общей системы отсчета. Это особенно полезно для проверки согласованности между результатами FSV и визуальной оценкой. Во-вторых, она позволяет перевести

результаты FSV на визуальную основу.

Валидация результатов использования методов электромагнитного моделирования, особенно в отношении экспериментальных результатов, часто включает в себя структурно сложные наборы данных. Количественное определение этих сравнений обычно делается «на глаз». Субъективность данного подхода затрудняет определение степени сходства. Кроме того, отсутствие шкалы, на основе которой должны быть проведены сравнения, делает оценку пар данных, в зависимости от их подобия, сложной задачей. Применение VRS обеспечивает некоторые гарантии, которых не хватает в визуальном подходе.

VRS представляет собой шеститочечную шкалу, для которой требуется двоичное решение на каждом узле. Алгоритм показан на рисунке 1.18. Метод FSV устраняет субъективность при валидации наборов данных, указывая на сравнения, которые должны быть выполнены. Таким образом, VRS определяет общий набор значений, на естественном языке, для сравнения наборов данных: «отлично», «очень хорошо», «хорошо», «умеренно», «плохо» и «очень плохо».

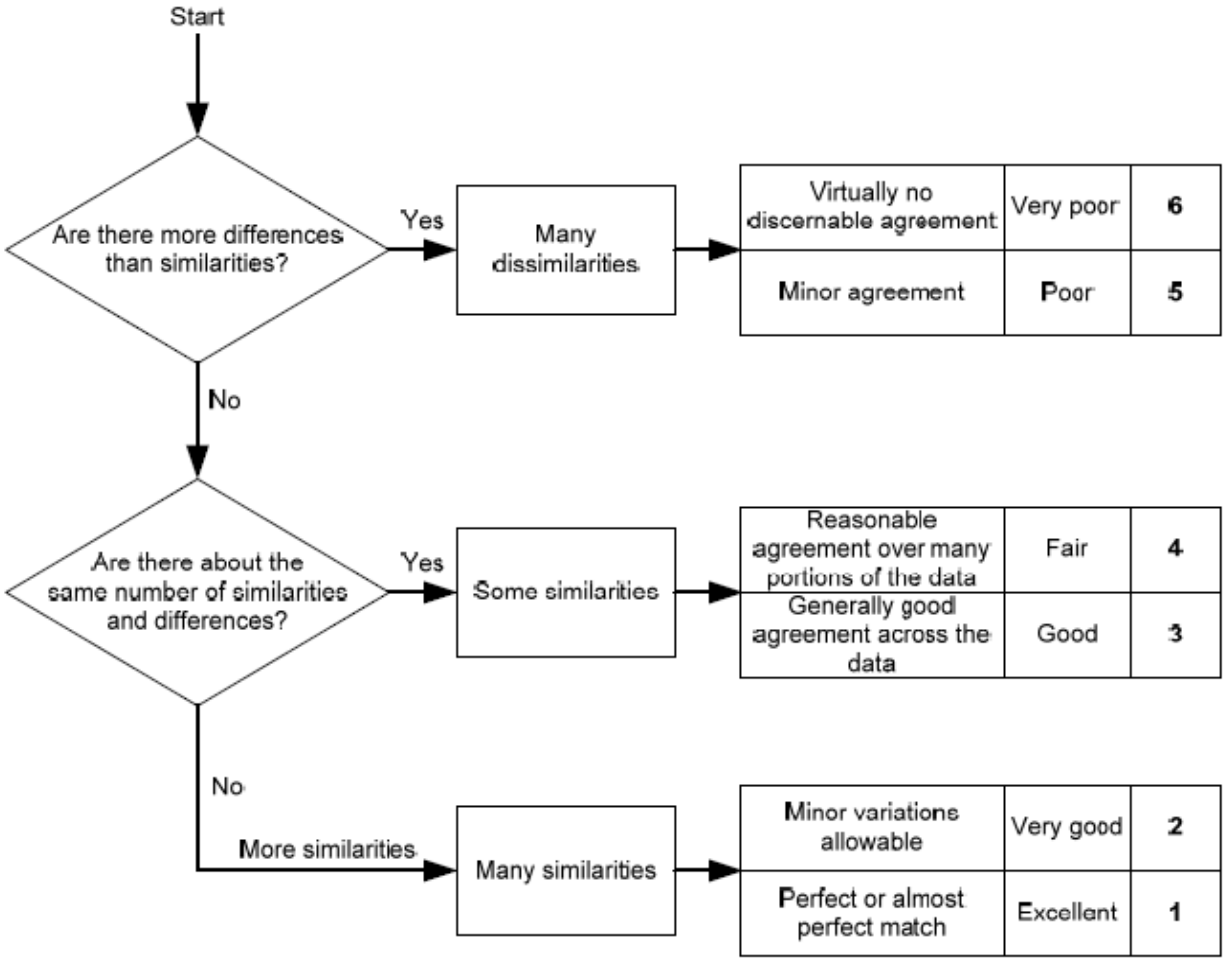


Рисунок 1.18 – VRS для визуального тестирования

В следующем разделе на примере тестовых структур приведены результаты валидации результатов моделирования с помощью использования нескольких программных продуктов, а также физического моделирования (без использования VRS).

1.7 Инструментальные средства

В данном подразделе приведены результаты моделирования тестовых структур: несимметричный вибратор (НВ) на идеально проводящей пластине, металлический корпус с двумя щелями и корпус блока питания (БП), возбужденные несимметричным вибратором. Далее приведены результаты компьютерного моделирования данных структур в системах FEKO и EMC Studio. Параметры рабочей станции, на которой производились

вычисления: процессор Intel Core Duo 1,66 ГГц, ОЗУ 1,5 Гб

1.7.1 Несимметричный вибратор на идеально проводящей пластине

Далее приведены результаты моделирования НВ (длина 12 см, радиус 0,5 см), который расположен в центре идеально проводящей пластины, размерами 40 см×30 см, рисунок 1.19. Целью моделирования являлось определение частотных зависимостей модуля входного импеданса и модуля коэффициента отражения НВ, в диапазоне от 200 МГц до 1 ГГц.

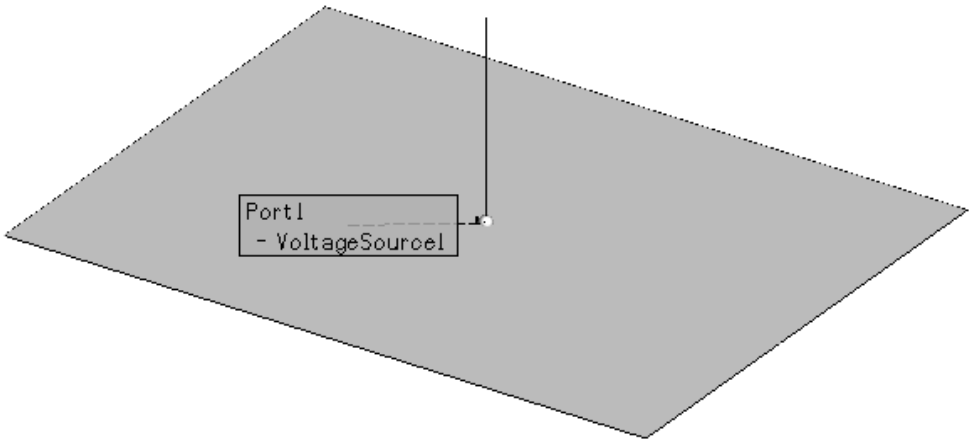
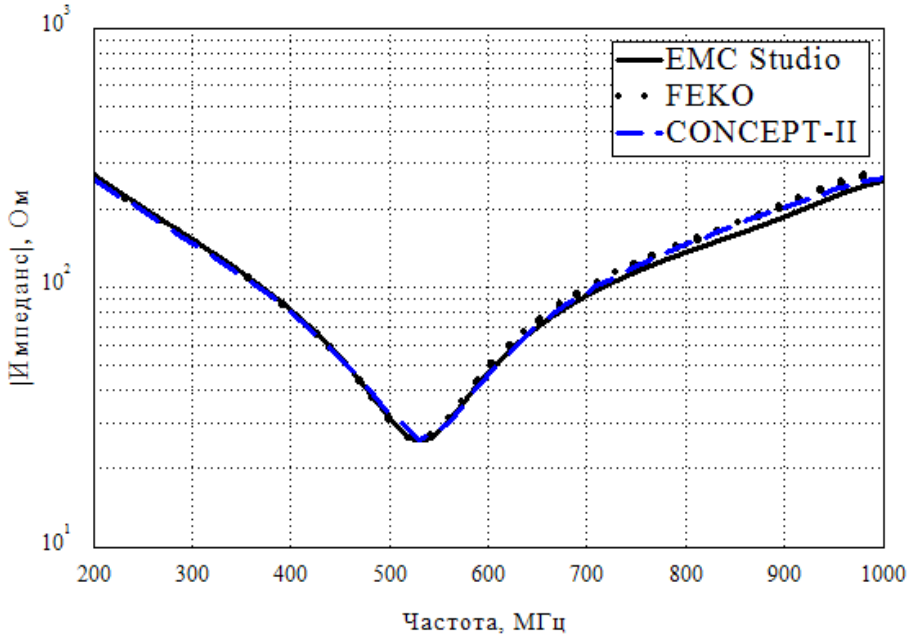
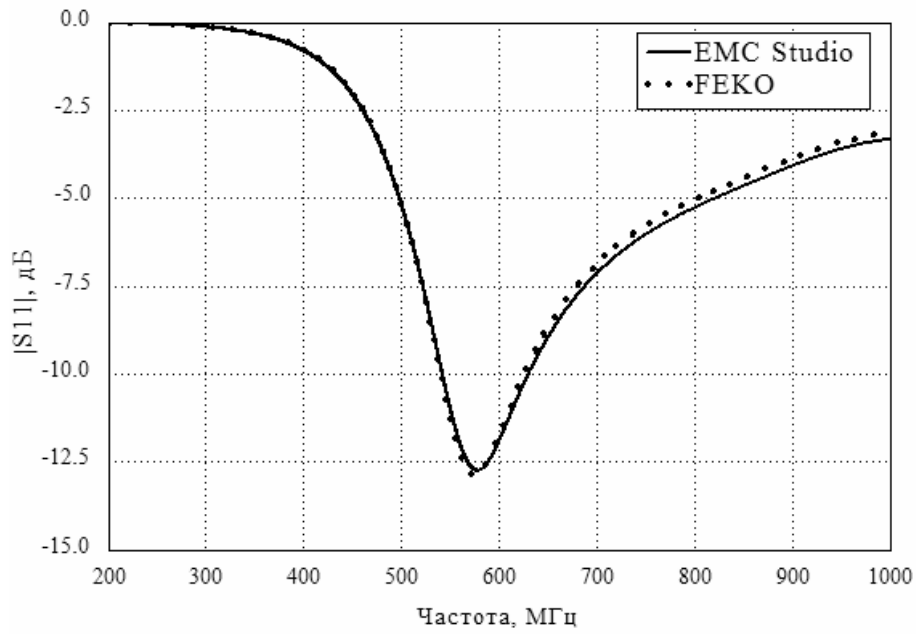


Рисунок 1.19 – НВ на идеально проводящей пластине

Далее приведены полученные частотные зависимости модуля входного импеданса (рисунок 1.20 *а*) и модуля коэффициента отражения (рисунок 1.20 *б*) НВ, полученные в рассматриваемых системах (кроме модуля коэффициента отражения в системе CONCEPT-II). При использовании системы FEKO поверхность пластины была разбита на 344 треугольника, а НВ на 9 сегментов. Время моделирования составило 38 секунд. В случае использования EMC Studio поверхность пластины была разбита на 344 треугольника, а НВ на 9 сегментов. Время вычисления в этом случае составило 17 секунд. При использовании системы CONCEPT-II поверхность пластины была разбита на 300 прямоугольников, а НВ на 9 сегментов. Время вычисления в этом случае составило 10 секунд.



а



б

Рисунок 1.20 – Зависимость модуля входного импеданса НВ от частоты (*а*) и зависимость модуля коэффициента отражения НВ от частоты (*б*)

Как видно из представленных зависимостей системы показали практически одинаковые результаты. Наибольшее расхождение во всем диапазоне составили: 20 Ом для входного импеданса на частоте 900 МГц, а для коэффициента отражения на 0,1 дБ на частоте 900 МГц.

1.7.2 Корпус с двумя щелями

В данном разделе рассмотрен пример моделирования металлического корпуса (480 мм×400 мм×300 мм) со щелями (200 мм×10 мм, расстояние от переднего края – 80 мм и 240 мм), который возбужден НВ (длиной 120 мм и радиусом 5 мм), рисунок 1.21. Задачей являлось определение частотных зависимостей модуля и фазы входного импеданса НВ, в диапазоне от 200 МГц до 1 ГГц.

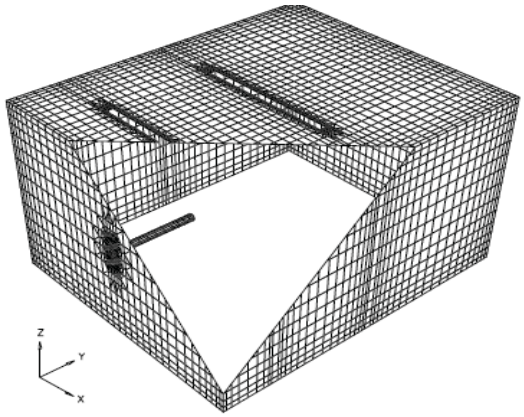


Рисунок 1.21 – Конфигурация металлического корпуса со щелями, возбужденного несимметричным вибратором

При использование системы FEKO поверхность корпуса была разбита на 1286 треугольников, а НВ на 9 сегментов. Время вычисления составило 14 минут. При использовании системы EMC Studio поверхность была разбита на 1280 треугольников, а НВ на 9 сегментов. Время вычисления составило 12 минут 49 сек.

Далее на рисунках приведены полученные частотные зависимости модуля входного импеданса (рисунок 1.22 *а*) и модуля коэффициента отражения (рисунок 1.22 *б*) НВ. Как видно из представленных зависимостей системы показали практически одинаковые результаты (как и на предыдущем примере). Наибольшее расхождение составили: 100 Ом для входного импеданса на частоте 880 МГц, а для коэффициента отражения 3 дБ на частоте 929 МГц.

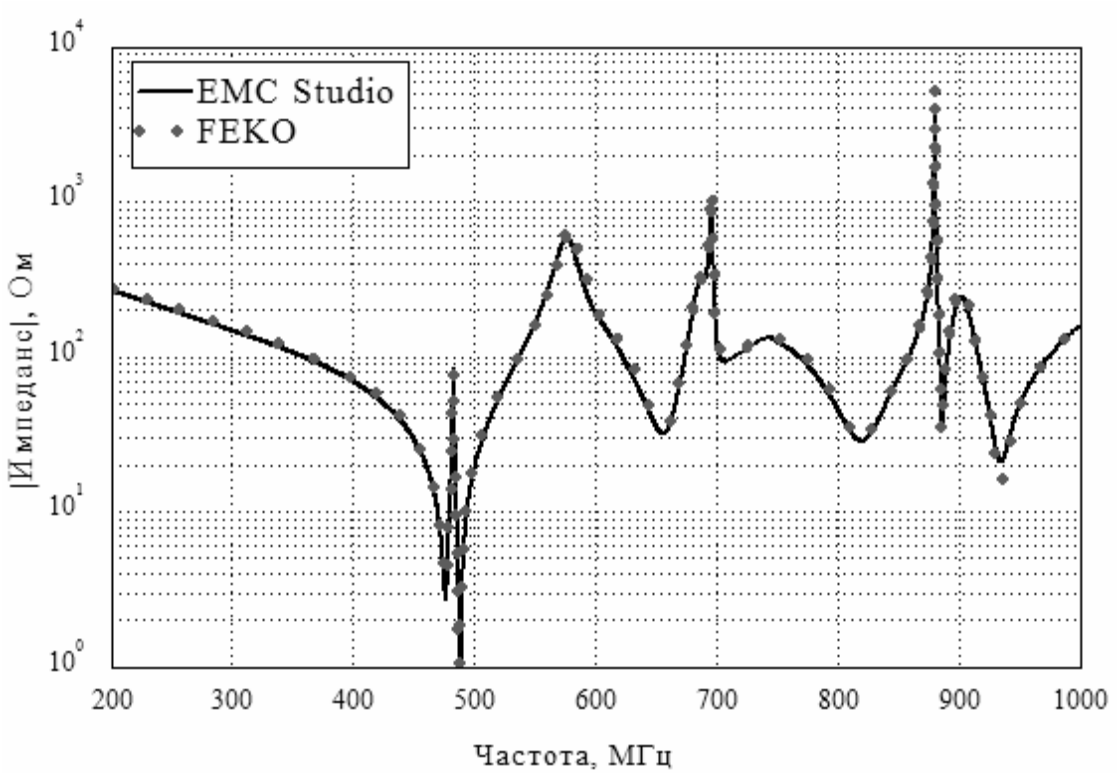


Рисунок 1.22 – Зависимость модуля входного импеданса НВ от частоты (*а*) и зависимость модуля коэффициента отражения НВ от частоты (*б*)

1.7.3 Корпус блока питания

В данном подразделе произведены результаты электродинамического моделирования корпуса БП, показанного на рисунке 1.23, возбужденного

несимметричным вибратором (радиусом 0,07 см), установленным в центре дна корпуса БП. Размеры корпуса приведены в приложении В. Материалом корпуса БП, являлся алюминий (проводимость $3{,}816\times10^{7}$ См/м), а НВ медь (проводимость $5{,}813\times10^{7}$ См/м). Целью моделирования являлось получение зависимости модуля коэффициента отражения НВ от частоты, в диапазоне от 300 МГц до 1,4 ГГц, для трех длин (12 см, 6 см и 3 см) НВ.

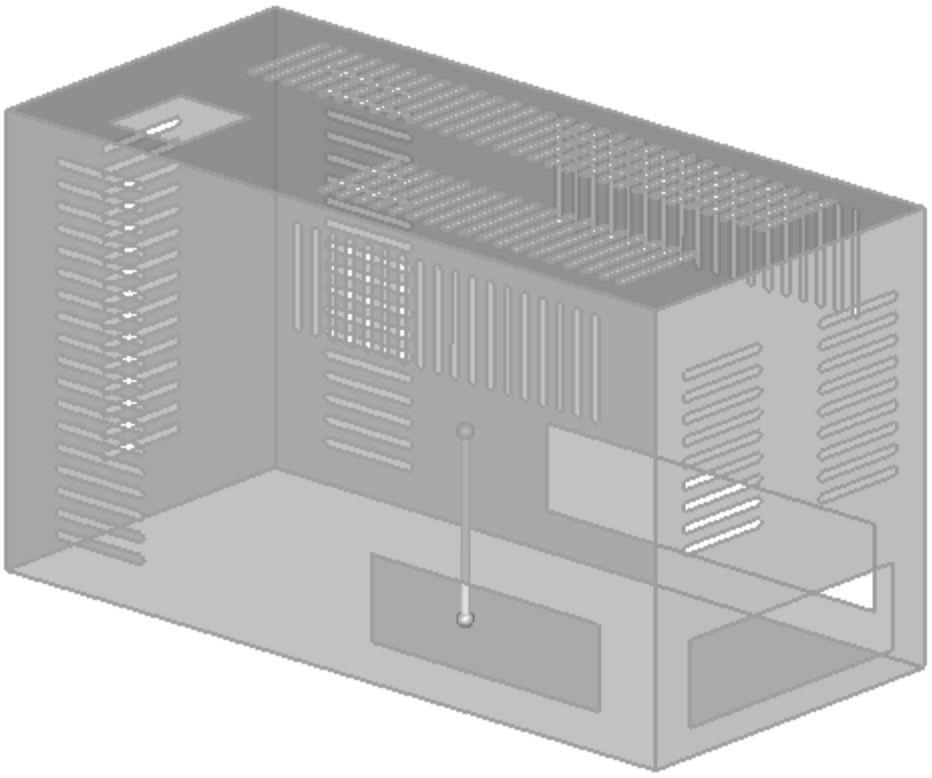


Рисунок 1.23 – Корпус БП

1.7.3.1 Длина вибратора 12 см

При использовании в данном случае системы FEKO поверхность корпуса БП была разбита на 8446 треугольников, а НВ на 20 сегментов. Время вычисления составило 4 часа 30 минут. В случае с EMC Studio поверхность корпуса БП была разбита на 5154 треугольника, а НВ на 13 сегментов. Время вычисления составило 2 часа 35 минут.

На рисунке 1.24приведены полученные частотные зависимости модуля коэффициента отражения НВ. Как видно из представленных зависимостей системы показали практически одинаковые результаты (как и на предыдущем

примере). Наибольшее расхождение составляет не более 0,1 дБ на частоте 960 МГц.

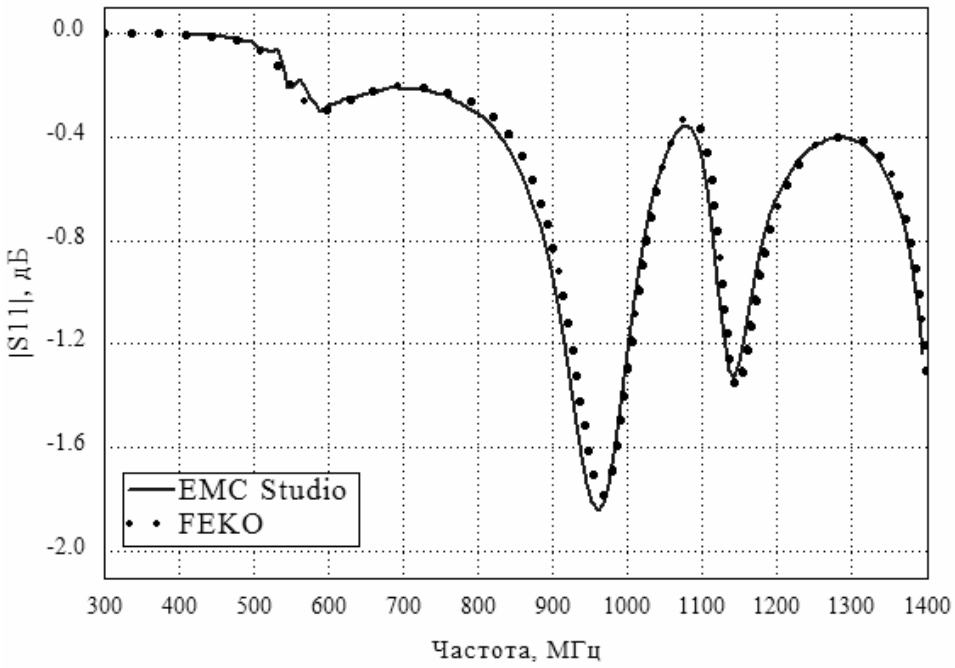


Рисунок 1.24 – Зависимость модуля коэффициента отражения НВ (длина 12 см) от частоты

1.7.3.2 Длина вибратора 6 см

При моделировании с использованием системы FEKO. Поверхность корпуса БП была разбита на 5126 треугольников и на 10411 треугольников, а НВ на 10 сегментов. Время вычисления в первом случае составило 2 часа 02 минуты, а во втором случае моделирование проводилось на более мощной рабочей станции. В случае с EMC Studio поверхность корпуса БП была разбита на 5120 треугольников, а НВ на 10 сегментов. Время вычисления составило 2 часа 16 минут, моделирование с более мелкой дискретизацией в EMC Studio произвести не удалось, т.к. лицензия от производителя идет только для конкретной рабочей станции.

Далее на рисунке 1.25 приведены полученные частотные зависимости модуля коэффициента отражения НВ. Видно, что системы показали практически идентичные результаты при одинаковой дискретизации. Наибольшее расхождение составляет не более 1 дБ в диапазоне частот от

1100 МГц до 1200 МГц. Однако, при уменьшении величины дискретизации в два раза, в диапазоне частот от 1100 МГц до 1200 МГц, остался только один резонанс на частоте 1185 МГц, на котором коэффициент отражения составляет – 16,5 дБ, в то же время общий вид характеристики сдвинулся вверх по частоте в среднем на 50 МГц.

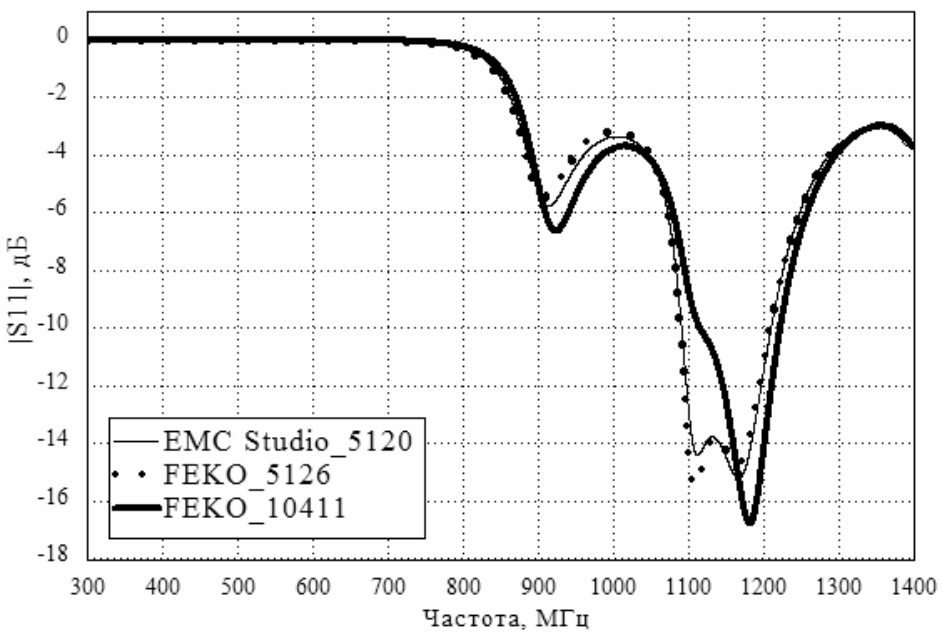


Рисунок 1.25 – Зависимость модуля коэффициента отражения НВ (длина 6 см) от частоты

1.7.3.3 Длина вибратора 3 см

При использовании в данном случае системы FEKO поверхность корпуса БП была разбита на 10411 треугольников, а НВ на 10 сегментов. В случае с EMC Studio поверхность корпуса БП была разбита на 5154 треугольника, а НВ на 6 сегментов. Время вычисления составило 2 часа 35 минут.

Далее на рисунке 1.26 приведены полученные частотные зависимости модуля коэффициента отражения НВ. Как и ранее системы показали практически одинаковые результаты. Наибольшее расхождение составляет не более 0,1 дБ на частоте 1125 МГц.

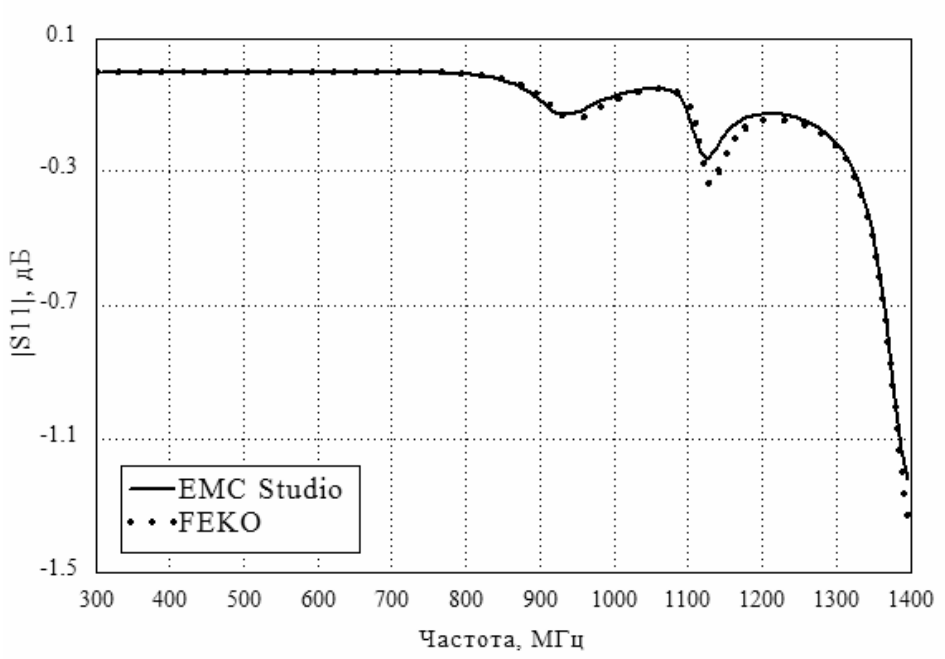


Рисунок 1.26 – Зависимость модуля коэффициента отражения НВ (длина 3 см) от частоты

1.7.3.4 Сравнение полученных результатов

В данном подразделе представлены результаты экспериментального моделирования и проведено сравнение с результатами компьютерного моделирования. В качестве тестовых структур были выбраны следующие конфигурации: металлический корпус с двумя щелями и корпус БП. Для корпуса с двумя щелями сравнение полученных результатов компьютерного моделирования в системах FEKO и EMC Studio предполагает сравнение с результатами измерений зарубежных исследователей. Для корпуса БП экспериментальное моделирование предполагает его макетирование и экспериментальное определение электрических параметров на макете.

1.7.3.5 Экспериментальное моделирование корпуса блока питания

Цель моделирования – получение частотных зависимостей модуля коэффициента отражения ($|S_{11}|$) для трех длин НВ, находящегося в корпусе БП.

Эксперимент был проведен в двух лабораториях на приборах «Обзор – 102» и «Обзор – 103». «Обзор – 103» позволяет проводить измерения в частотном диапазоне от 0,3 МГц до 1,4 ГГц, «Обзор – 102» отличается тем, что

позволяет проводить измерения в частотном диапазоне от 0,3 МГц до 1,2 ГГц, далее приводится описание прибора «Обзор – 103».

«Обзор – 103» – измеритель комплексных коэффициентов передачи (ИККП «Обзор – 103»), предназначенный для использования во время проверки, настройки и разработки различных радиотехнических устройств в условиях промышленного производства и лабораторий, в том числе в составе автоматизированных измерительных стендов. ИККП «Обзор – 103» состоит из измерительного блока, измерительных секций, персонального компьютера под управлением ОС «WINDOWS» и дополнительных устройств, обеспечивающих функционирование прибора, рисунок 1.27. Связь измерительного блока с персональным компьютером осуществляется через USB интерфейс.

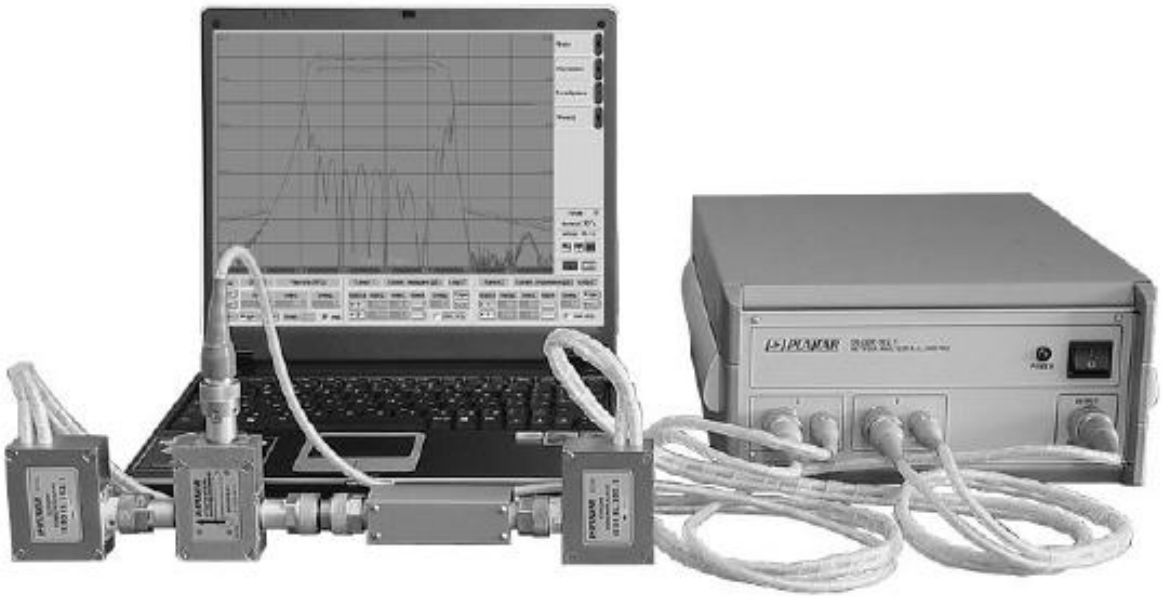


Рисунок 1.27 – Вид ИККП «Обзор – 103»

Вид тестовой структуры, показанный на рисунке 1.28, представляет собой НВ, размещенный на дне корпуса БП (верхняя крышка БП удалена, для того чтобы видеть внутреннюю конфигурацию макета). Вибратор показан на рисунке 1.29 и представляет собой медный провод радиусом 0,07 см. Соединение измерительного комплекса с тестовой структурой осуществляется через разъем SMA, центральный вывод которого припаян к НВ, а внешней частью соединен с корпусом БП посредством резьбы.

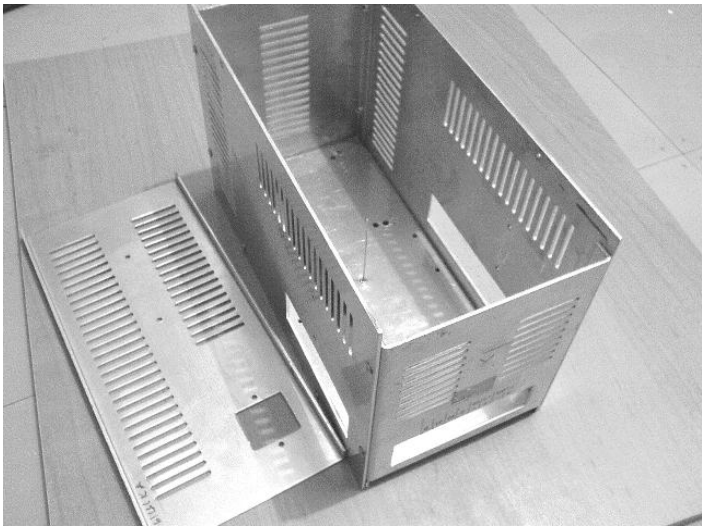


Рисунок 1.28 – Вид тестовой структуры

Рисунок 1.29 – Вид НВ и разъема SMA

Далее приведена схема измерения комплексного коэффициента отражения S_{11}, рисунок 1.30.

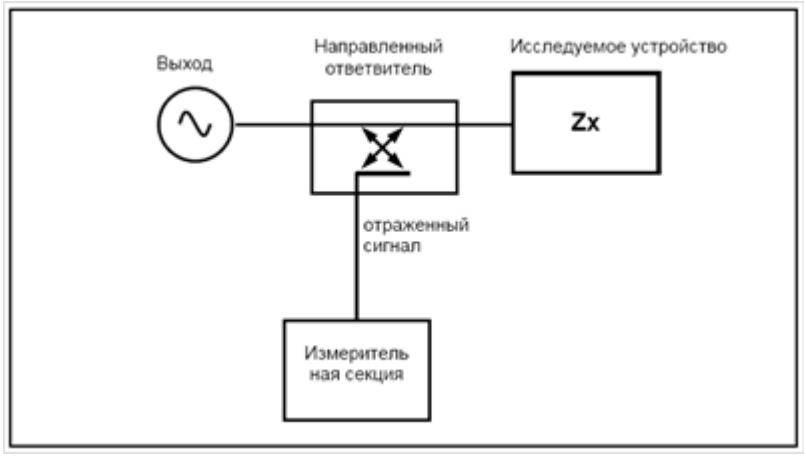


Рисунок 1.30 – Схема для измерения комплексного коэффициента отражения S_{11}

Принцип измерения комплексных коэффициентов передачи заключается в подаче на исследуемое устройство испытательного сигнала на заданной частоте, последующего измерения амплитуды и фазы, прошедших и

отраженных исследуемым устройством сигналов и сравнения их с амплитудой и фазой испытательного сигнала. Комплексный коэффициент отражения S_{11} определяется как отношение напряжения отраженного сигнала к напряжению падающего сигнала в комплексном виде.

Измеряемые S-параметры доступны для отображения в различных форматах, принятых для тех или иных видов измерений (двумерный график, диаграмма Смита, текстовый файл и др.). Выбор формата представления S-параметров осуществляется в ПО с помощью специальной панели управления форматом.

На рисунке 1.31 представлены частотные зависимости модуля коэффициента отражения НВ, длиной 12 см, полученные путем измерений и компьютерного моделирования.

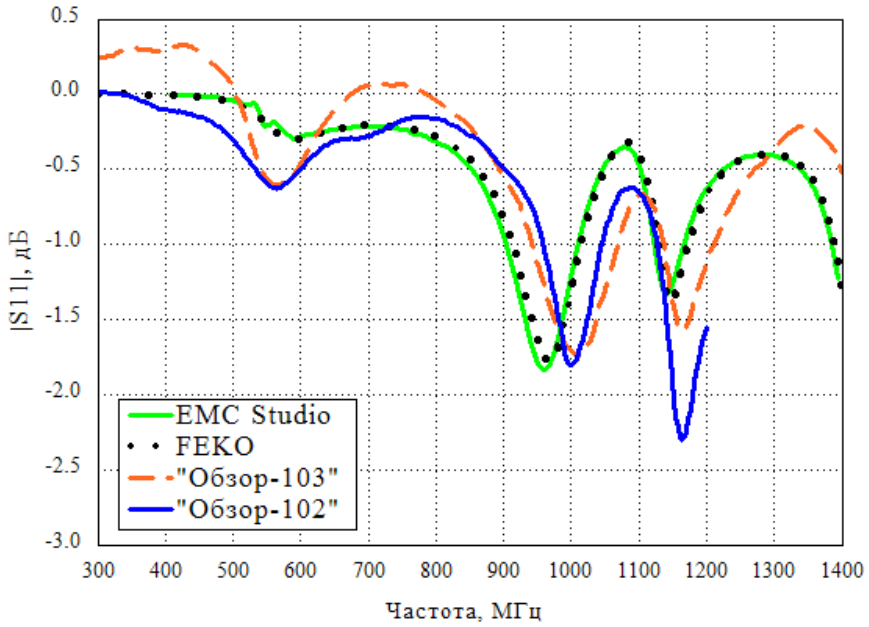


Рисунок 1.31 – Зависимости модуля коэффициента отражения НВ (длина 12 см) от частоты

Изменение характеристик по уровню происходит в узких пределах от 0,5 до –2,5 дБ, что близко к полному отражению энергии от НВ. Общий вид характеристик идентичен. Расхождения между измерениями во всем частотном диапазоне по уровню составляет не более 0,7 дБ, расхождение между моделированием и измерениями, сдвиг по частоте, составляет не более 50 МГц.

На рисунке 1.32 представлены частотные зависимости модуля коэффициента отражения НВ длиной 6 см, полученные путем измерений и компьютерного моделирования.

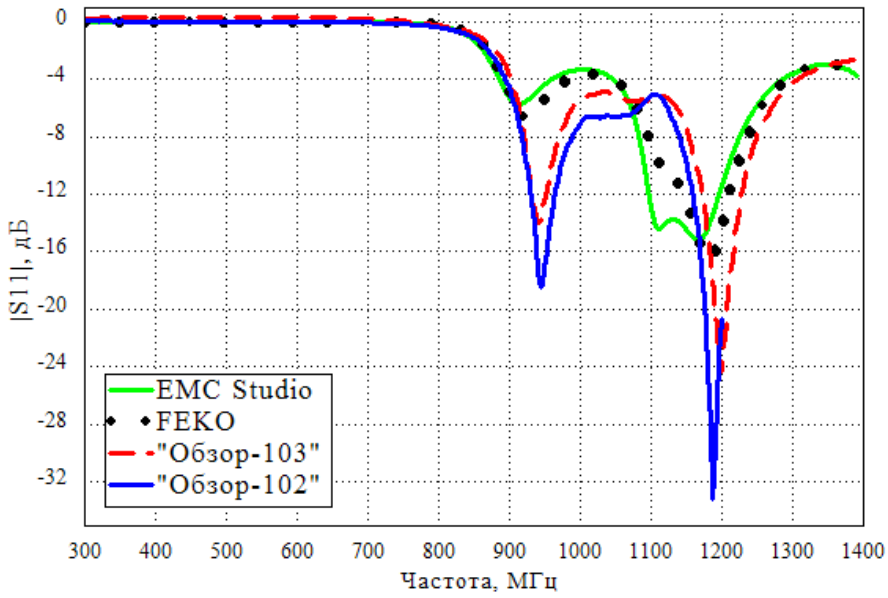


Рисунок 1.32 – Зависимости модуля коэффициента отражения НВ (длина 6 см) от частоты

Из результатов эксперимента видно, что на частоте, близкой к 1200 МГц (1185 МГц), НВ имеет хорошее согласование с нагрузкой, т.е. практически вся энергия, поступающая от генератора передается в антенну, об этом говорит $|S_{11}| = -24$ дБ (– 32 дБ). Данная частота близка к резонансной частоте четверть волнового НВ, находящегося на бесконечной идеально проводящей плоскости. Результаты моделирования с более мелкой дискретизацией идентичны измерениям, однако, наблюдается различие на резонансе (1185 МГц) по уровню на 16 дБ. Результаты моделирования с более крупной дискретизацией показывают согласование антенны с генератором в полосе 1100 – 1185 МГц, чего не наблюдается в измерениях.

На рисунке 1.33 представлены частотные зависимости модуля коэффициента отражения НВ, длиной 3 см, полученные путем измерений и компьютерного моделирования (пункт 1.7.3.3).

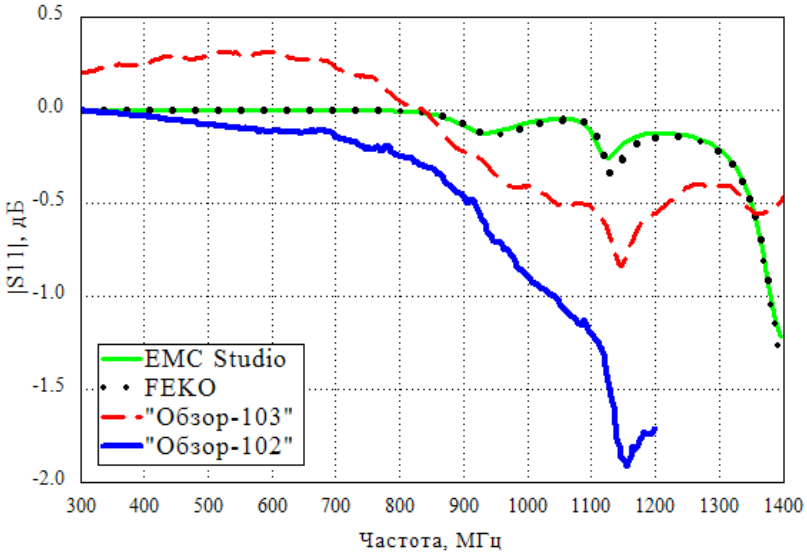


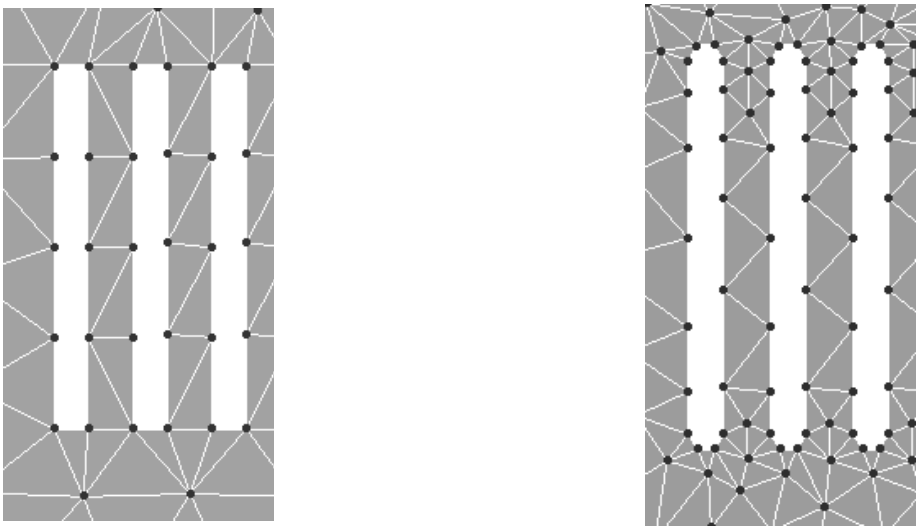
Рисунок 1.33 – Зависимости модуля коэффициента отражения несимметричного вибратора (длина 3 см) от частоты

Изменение характеристик по уровню происходит в узких пределах от 0,5 до –2 дБ, что близко к полному отражению энергии от НВ. Общий вид характеристик идентичен. Наибольшее расхождения между измерениями по уровню составляет 1 дБ на частоте 1150 МГц, расхождение между моделированием и измерениями по уровню составляет 1,5 дБ.

Причиной различий между результатами измерений на приборах «Обзор – 102» и «Обзор – 103» является то, что эксперимент проводился в разных лабораториях, соответственно в разных электромагнитных обстановках. Так, видно, что модуль коэффициента отражения НВ на приборе «Обзор – 103» больше нуля (шкала в дБ), физически это означает, что отраженный сигнал от нагрузки (НВ) больше входного на 0,3 дБ на нижних частотах исследуемого диапазона. Это вызвано внешними наводками на НВ, которых нет в компьютерном моделировании, в последнем случае структура находится в свободном пространстве.

Сравнивая результаты компьютерного моделирования и экспериментов видно, что общий вид характеристик (число резонансов и значения по уровню) в среднем идентичен. Однако общая характеристика, в случае компьютерного моделирования, является сдвинутой ниже по частоте на 50 МГц – 100 МГц относительно характеристики измерений (рисунок 1.31 – рисунок 1.33).

Оценим влияние величины дискретизации на результаты моделирования. Данный корпус БП геометрически сложен, т.к. имеет большое количество щелей и вырезов, причем щели имеют закругленную форму. На рисунке 1.34 представлен участок поверхности корпуса БП с разной дискретизацией: грубая дискретизация (поверхность корпуса БП разбита на 5126 треугольников) – *а*, учащенная дискретизация (поверхность корпуса БП разбита на 10411 треугольников) – *б*.



а *б*

Рисунок 1.34 – Участок поверхности корпуса БП с крупной дискретизацией (*а*), со средней дискретизацией (*б*)

Осуществить более мелкую дискретизацию поверхности корпуса БП в программных пакетах для более точного описания геометрии не представилось возможным, из-за нехватки объема ОЗУ на рабочей станции, ниже приведен график зависимости требуемого объема ОЗУ от величины дискретизации поверхности на плоские треугольники, рисунок 1.35.

Рисунок 1.35 – Зависимость требуемого объема ОЗУ
от величины дискретизации поверхности

Недостаточная дискретизация, является основной причиной расхождения результатов компьютерного моделирования и измерений. Кроме того, необходимо отметить и другие возможные причины:

- измерения проводились не в безэховой камере, а в неопределенной электромагнитной обстановке, в которой присутствуют внешние наводки и отражения от конструкций лаборатории;
- в реальности металл корпуса БП имеет неоднородную проводимость по всей поверхности;
- провод, которым представлен НВ, в реальности имеет искривления, которых нет в моделировании;
- неточность задание размеров корпуса БП при компьютерном моделировании.

2 Лабораторные работы

2.1 Метод прогонки

Цель работы – получение навыков решения СЛАУ с трехдиагональной матрицей методом прогонки.

Для достижения поставленной цели используется программное обеспечение GNU Octave или Scilab.

2.1.1 Краткая теоретическая справка

Рассмотрим метод прогонки – простой и эффективный алгоритм решения СЛАУ с трехдиагональными матрицами вида

$$\begin{array}{ccccccccccc}
b_1x_1 & + & c_1x_2 & & & & & & & = & d_1, \\
a_2x_1 & + & b_2x_2 & + & c_2x_3 & & & & & = & d_2, \\
 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \\
 & a_ix_{i-1} & + & b_ix_i & + & c_ix_{i+1} & & & & = & d_i, \\
 & & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \\
 & & & a_{N-1}x_{N-2} & + & b_{N-1}x_{N-1} & + & c_{N-1}x_N & = & d_{N-1}, \\
 & & & & & a_Nx_{N-1} & + & b_Nx_N & = & d_N.
\end{array} \qquad (2.1)$$

СЛАУ такого вида, как будет показано ниже, часто возникают при решении прикладных задач. Преобразуем первое уравнение (2.1) к виду

$$x_1 = \alpha_1x_2 + \beta_1, \qquad (2.2)$$

где $\alpha_1 = -c_1 / b_1$, $\beta_1 = d_1 / b_1$. Подставим выражение для x_1 во второе уравнение системы:

$$a_2(\alpha_1x_2 + \beta_1) + b_2x_2 + c_2x_3 = d_2.$$

Преобразуем это уравнение к виду

$$x_2 = \alpha_2x_3 + \beta_2, \qquad (2.3)$$

где $\alpha_2 = -c_2 / (b_2 + a_2\alpha_1)$, $\beta_2 = (d_2 - a_2\beta_1) / (b_2 + a_2\alpha_1)$.

Выражение (2.3) подставим в третье уравнение системы и т.д. На i-м шаге этого процесса ($1 < i < N$) i-е уравнение системы преобразуется к виду

$$x_i = \alpha_ix_{i+1} + \beta_i, \qquad (2.4)$$

где $\alpha_i = -c_i / (b_i + a_i\alpha_{i-1})$, $\beta_i = (d_i - a_i\beta_{i-1}) / (b_i + a_i\alpha_{i-1})$. На N-м шаге подстановка в последнее уравнение выражения $x_{N-1} = \alpha_N x_{N+1} + \beta_{N-1}$ дает:

$$a_N(\alpha_{N-1}x_N + \beta_{N-1}) + b_N x_N = d_N.$$

Откуда можно определить значение x_N:

$$x_N = \beta_N = (d_N - a_N\beta_{N-1}) / (b_N + a_N\alpha_{N-1}).$$

Значения остальных неизвестных x_i для $i = N-1,\ N-2,\ \ldots,\ 1$ теперь легко вычисляются по формуле (2.4). Сделанные преобразования позволяют организовать вычисления метода прогонки в два этапа.

Прямой ход метода прогонки (прямая прогонка) состоит в вычислении прогоночных коэффициентов α_i $(1 \le i < N)$ и β_i $(1 \le i < N)$. При $i = 1$ коэффициенты вычисляют по формулам

$$\alpha_1 = -c_1 / \gamma_1,\ \beta_1 = d_1 / \gamma_1,\ \gamma_1 = b_1, \tag{2.5}$$

а при $i = 2, 3, \ldots, N-1$ – по рекуррентным формулам

$$\alpha_i = -c_i / \gamma_i,\ \beta_i = (d_i - a_i\beta_{i-1}) / \gamma_i,\ \gamma_i = b_i + a_i\alpha_{i-1}. \tag{2.6}$$

При $i = N$ прямая прогонка завершается вычислениями

$$\beta_N = (d_N - a_N\beta_{N-1}) / \gamma_N,\ \gamma_N = b_N + a_N\alpha_{N-1}. \tag{2.7}$$

Обратный ход метода прогонки (обратная прогонка) дает значения неизвестных. Сначала полагают $x_N = \beta_N$. Затем значения остальных неизвестных вычисляют по формуле

$$x_i = \alpha_i x_{i+1} + \beta_i,\ i = N-1, N-2, \ldots, 1. \tag{2.8}$$

Вычисления ведут в порядке убывания значений i от $N-1$ до 1.

Непосредственный подсчет показывает, что для реализации вычислений по формулам (2.5) – (2.8) требуется примерно $8N$ арифметических операций, тогда как в методе Гаусса это число составляет примерно $(2/3)N^3$. Важно и то, что трехдиагональная структура матрицы системы позволяет использовать для ее хранения $3N - 2$ машинных слова.

Таким образом, при одной и той же производительности и оперативной памяти компьютера метод прогонки позволяет решать системы гораздо большей размерности, чем стандартный метод Гаусса для систем уравнений с

плотной (заполненной) матрицей.

Описанный вариант метода прогонки можно рассматривать как одну из схем метода Гаусса (без выбора главного элемента), в результате прямого хода которого исходная трехдиагональная матрица

$$\mathbf{A}=\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_N & b_N \end{pmatrix}$$

представляется в виде произведения двух двухдиагональных матриц:

$$\mathbf{A} = \mathbf{LU}, \tag{2.9}$$

где

$$\mathbf{L} = \begin{pmatrix} \gamma_1 & 0 & 0 & \dots & 0 & 0 \\ a_2 & \gamma_2 & 0 & \dots & 0 & 0 \\ 0 & a_3 & \gamma_3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_N & \gamma_N \end{pmatrix}, \mathbf{U} = \begin{pmatrix} 1 & -\alpha_1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -\alpha_2 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & -\alpha_{N-1} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix},$$

Так как для определения **L** и **U** нет необходимости вычислять коэффициенты β_i, то общее число операций на получение разложения (2.9) составляет порядка $3N$.

2.1.2 Порядок выполнения работы

1. Программно реализовать решение СЛАУ метод прогонки.
2. Продемонстрировать правильность реализации на примере решения СЛАУ

$$\begin{aligned} 5x_1 - x_2 &= 2{,}0 \\ 2x_1 + 4{,}6x_2 - x_3 &= 3{,}3 \\ 2x_2 + 3{,}6x_3 - 0{,}8x_4 &= 2{,}6 \\ 3x_3 + 4{,}4x_4 &= 7{,}2 \end{aligned}.$$

Система имеет решение: $x_1 = 0{,}5256$, $x_2 = 0{,}628$, $x_3 = 0{,}64$, $x_4 = 1{,}2$.

3. Получить аналитические оценки вычислительной сложности разработанного алгоритма.

4. Оценить вычислительные затраты (время вычисления, требуемая память) на решение СЛАУ при N = 1000, 2000, 5000.

5. Проанализировать полученные результаты и сформулировать выводы по работе.

6. Оформить отчет.

2.1.3 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

2.2 Блочное LU-разложение

Цель работы – получение навыков многократного решения СЛАУ с использованием блочного LU-разложения.

Для достижения поставленной цели используется программное обеспечение GNU Octave или Scilab.

2.2.1 Краткая теоретическая справка

В работе изучается вопрос многократного решения СЛАУ с изменяющейся матрицей и неизменной правой частью вида $\mathbf{S}_k\mathbf{\Sigma}_k = \mathbf{V}$, $k = 1, 2, \ldots, m$. Размеры матриц: $\mathbf{S} - N \times N$, $\mathbf{\Sigma}$ и $\mathbf{V} - N \times N_{\text{COND}}$.

В настоящее время в большинстве систем компьютерного моделирования для решения СЛАУ с плотной матрицей применяются прямые методы, такие как метод исключения Гаусса или его версия, основанная на LU-разложении исходной матрицы. Вычислительные затраты данных методов пропорциональны N^3, что неприемлемо при необходимости многократных

вычислений с изменяющейся матрицей. В таком случае подходящим методом для решения СЛАУ является блочная версия LU-разложения с последующим решением СЛАУ. В отличие от обычного алгоритма LU-разложения в данной версии матрицы **L** и **U** вычисляются не поэлементно, а целыми блоками. Поэтому нет необходимости каждый раз выполнять полное LU-разложение матрицы СЛАУ, а надо пересчитывать только блок, соответствующий изменившимся элементам исходной матрицы. Этот прием может значительно ускорить вычисления.

Для оценки эффективности применения блочного LU-разложения рассмотрим его алгоритм на примере матрицы **S**, представленной в виде

$$\mathbf{S} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

где блоки размерами: **A** – $N_C \times N_C$, **B** – $N_C \times N_D$, **C** – $N_D \times N_C$, **D** – $N_D \times N_D$, $N = N_C + N_D$. Для разложения необходимо:

1. Присвоить $\mathbf{U}_{11} = \mathbf{A}$, $\mathbf{U}_{12} = \mathbf{B}$.
2. Вычислить $\mathbf{L}_{21} = \mathbf{C}\mathbf{U}_{11}^{-1}$.
3. Вычислить $\mathbf{U}_{22} = \mathbf{D} - \mathbf{L}_{21}\mathbf{U}_{12}$.

Далее рассмотрим частный случай многократного изменения диагональных элементов блока **D** (рисунок 2.1). В данном случае при использовании LU-разложения получим следующий алгоритм (далее Алгоритм 1).

Алгоритм 1: *m*-кратное решения СЛАУ с помощью LU-разложения

```
1       Для k от 1 до m
2           Если k=1
3               Вычислить элементы матрицы S_1
4               Вычислить элементы матрицы воздействий V
5               Решить СЛАУ S_1Σ_1 = V с помощью LU-разложения
6           Иначе
7               Сформировать матрицу S_k (изменить диагональные элементы
                блока D)
8               Решить СЛАУ S_kΣ_k = V с помощью LU-разложения
9       Увеличить k
```

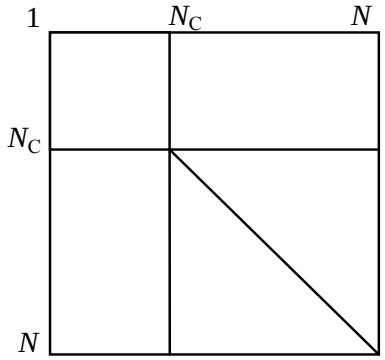


Рисунок 2.1 – Структура матрицы S
при изменении диагональных элементов блока **D**

Далее приведена программная реализация Алгоритма 1, используемая в работе:

```
N=10;
m=10;
Ncond=2;
Nd=N/2;
S=rand(N,N);
SIGMA=rand(N,Ncond);
V=rand(N,Ncond);
[l,u,p]=lu(S,'vector');
SIGMA = u\(l\(V(p,:)));
beg_=Nd*(N+1)+1;
step_=N+1;
end_=N*N;
norm(V-S*SIGMA)
for i=2:m
   S(beg_:step_:end_)=rand();
   [l,u,p]=lu(S,'vector');
   SIGMA = u\(l\(V(p,:)));
   norm(V-S*SIGMA)
end
```

При многократном изменении диагональных элементов блока **D** в блочном разложении изменяются только элементы блока $\mathbf{U}_{22}$, а все остальные блоки ($\mathbf{U}_{11}$, $\mathbf{U}_{12}$, $\mathbf{L}_{21}$) останутся неизменными. За счет этого достаточно лишь однократно вычислить $\mathbf{U}_{11}^{-1}$ и $\mathbf{L}_{21}\mathbf{U}_{12}$. Таким образом, при выполнении m вычислений будет однократно выполняться вычислительно затратное разложение матрицы СЛАУ (требующее приведения матрицы к пригодному для дальнейших вычислений

виду) и последующие $(m-1)$ вычислений блока $\mathbf{U}_{22}$. Далее приведен соответствующий алгоритм (далее называемый Алгоритм 2) для решения СЛАУ вида:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}_k \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_{0k} \\ \mathbf{\Sigma}_{1k} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_{0k} \\ \mathbf{V}_{1k} \end{bmatrix}, \begin{bmatrix} \mathbf{X}_0 \\ \mathbf{X}_1 \end{bmatrix} - \text{вспомогательная матрица.}$$

Размеры матриц: $\mathbf{\Sigma}_{0k}$, $\mathbf{V}_{0k}$, $\mathbf{X}_0 - N_C \times N_{\text{COND}}$, $\mathbf{\Sigma}_{1k}$, $\mathbf{V}_{1k}$, $\mathbf{X}_1 - N_D \times N_{\text{COND}}$.

Алгоритм 2: *m*-кратное решения СЛАУ с помощью блочного LU-разложения

1 Для k от 1 до m
2 Если k=1
3 Вычислить элементы матрицы $\mathbf{S}_1$
4 Сохранить элементы главной диагонали блока $\mathbf{D}$ в матрице $\mathbf{Diag}_1$
6 Вычислить элементы матрицы воздействий $\mathbf{V}$
7 $\mathbf{A} = \mathbf{A}^{-1}$
8 $\mathbf{B} = \mathbf{AB}$
9 $\mathbf{D} = \mathbf{D} - \mathbf{CB}$
10 $\mathbf{X}_0 = \mathbf{AV}_0$
11 $\mathbf{X}_1 = \mathbf{V}_1 - \mathbf{CX}_0$
12 $\mathbf{\Sigma}_{1k} = \mathbf{D}^{-1}\mathbf{X}_1$
13 $\mathbf{\Sigma}_{0k} = \mathbf{X}_0 - \mathbf{B\Sigma}_{1k}$
14 $\mathbf{D} = \mathbf{D} - \mathbf{Diag}_1$
15 Иначе
16 Вычислить элементы матрицы $\mathbf{Diag}_k$
17 $\mathbf{\Sigma}_{1k} = (\mathbf{D} + \mathbf{Diag}_k)^{-1}\mathbf{X}_1$
18 $\mathbf{\Sigma}_{0k} = \mathbf{X}_0 - \mathbf{B\Sigma}_{1k}$
19 Увеличить k

Для аналитического выражения ускорения рассмотрим отношение (β) общего времени решения m СЛАУ с помощью последовательного m-кратного LU-разложения ко времени с помощью блочного LU-разложения:

$$\beta = \frac{mT_{LU}}{T_1 + (m-1)T_S} \tag{2.1}$$

где T_1 – время первого решения, в которое входит обращение блока $\mathbf{U}_{11}$ размером $N_C \times N_C$ и последующее решение СЛАУ с нахождением вектора неизвестных; T_S – время вычисления блока $\mathbf{U}_{22} = \mathbf{D} - \mathbf{L}_{21}\mathbf{U}_{12}$ и последующего решения СЛАУ (время решения СЛАУ при готовом LU-разложении примем

равным для двух алгоритмов). Из (2.1) получим оценку максимально возможного ускорения:

$$\beta_{max} = \lim_{m\to\infty} \frac{mT_{LU}}{T_1 + (m-1)T_S} = \frac{T_{LU}}{T_S} \qquad (2.2)$$

Из (2.1)–(2.2) следует, что чем больше m, тем меньше ускорение зависит от времени первого решения. Также видно, что величина ускорения обратно пропорциональна времени вычисления блока $\mathbf{U}_{22}$, которое определяется порядком этого блока. Так, при больших m и малых N_D можно получить значительное ускорение многократных вычислений, в то время как при малых m и больших N_D ускорение будет незначительным или его вообще не будет.

2.2.2 Методические указания

При использовании в работе средств Scilab, из-за отсутствия аналогичной реализации, Алгоритм 1 реализуется с помощью функций [l,u]=lu(S) и SIGMA= u\(l\V).

2.2.3 Порядок выполнения работы

7. Разобрать работу программной реализации Алгоритма 1.
8. Программно реализовать Алгоритм 2.
9. Продемонстрировать правильность программной реализации Алгоритма 2 на примере решения двух СЛАУ

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & -10 \\ 1 & 3 & 5 & 7 \\ -2 & 4 & -6 & 8 \end{bmatrix} \begin{bmatrix} \sigma_1^1 \\ \sigma_2^1 \\ \sigma_3^1 \\ \sigma_4^1 \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 16 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & -10 \\ 1 & 3 & 4 & 7 \\ -2 & 4 & -6 & -3 \end{bmatrix} \begin{bmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \sigma_3^2 \\ \sigma_4^2 \end{bmatrix} = \begin{bmatrix} 10 \\ 8 \\ 16 \\ 4 \end{bmatrix}.$$

10. Выполнить оценку временных затрат на реализацию обоих алгоритмов при: N = 1000, 2000, 3000; $N_D = N/2$; N_{COND} = 1, 5, 10; m =1, 5, 10, 50, 100, 500. При вычислениях использовать произвольно заданные матрицы требуемого размера. Результаты свести в таблицу(ы).
11. Оценить полученное ускорение (β) – вычислительные оценки. Результаты свести в таблицу(ы).

12. Выполнить сравнение вычислительных (п. 5) и аналитических (выражения (2.1) и (2.2)) оценок ускорения.

13. Проанализировать полученные результаты и сформулировать выводы по работе.

14. Оформить отчет.

2.2.4 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

2.3 Метод сопряженных градиентов

Цель работы – получение навыков программной реализации метода сопряженных градиентов для итерационного метода решения СЛАУ.

Для достижения поставленной цели используется программное обеспечение GNU Octave или Scilab.

2.3.1 Краткая теоретическая справка

Исторически первые итерационные методы основывались на циклическом покомпонентном изменении вектора решения, осуществляемом таким образом, чтобы обнулить соответствующий коэффициент вектора невязки и тем самым уменьшить его норму (метод Якоби, Гаусса-Зейдела и др.). Подобная методика уточнения решения получила название релаксация. Хотя в настоящее время такие методы в их классической формулировке уже практически не применяются, существуют определенные классы задач, для которых разработаны их модификации, хорошо себя зарекомендовавшие. Кроме того эти методы могут быть применены не в качестве самостоятельного средства решения СЛАУ, а для предобусловливания.

На сегодняшний день самыми эффективными методами являются проекционные методы Крыловского типа (Krylov subspace methods). Идею таких подпространств в 1931 г. предложил Алексей Николаевич Крылов – русский кораблестроитель, специалист в области механики, математик. Одним из первых был разработан метод сопряженных градиентов. Метод предназначен для решения СЛАУ с симметричной матрицей. Иллюстрация работы метода при n=2 приведена на рисунок 2.2 ($\mathbf{x}_0$ – начальное приближение).

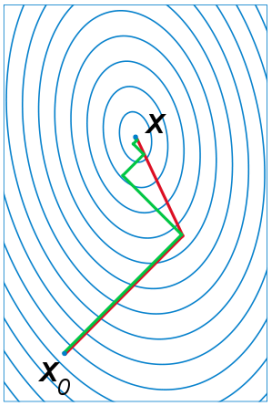


Рисунок 2.2 – Сравнение методов градиентного спуска (зеленый) и метода сопряженных градиентов для n=2

Далее приведен алгоритм метода сопряженных градиентов (CG).

Алгоритм метода CG

Выбрать начальное приближение $\mathbf{x}_0$
Вычислить $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\,\mathbf{x}_0$
Для $i = 1, 2, \ldots$ **до сходимости или до** N_{it}^{max}
    $\mathbf{z}_{i-1} = \mathbf{r}_{i-1}$
    $\rho_{i-1} = (\mathbf{r}_{i-1}, \mathbf{z}_{i-1})$
    Если $i = 1$
        $\mathbf{p}_1 = \mathbf{z}_0$
    Иначе
        $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
        $\mathbf{p}_i = \mathbf{z}_{i-1} + \beta_{i-1}\,\mathbf{p}_{i-1}$
    $\mathbf{q}_i = \mathbf{A}\,\mathbf{p}_i$
    $\alpha_i = \rho_{i-1} / (\mathbf{p}_i, \mathbf{q}_i)$
    $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\,\mathbf{p}_i$
    $\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i\,\mathbf{q}_i$
    Если $\|\mathbf{r}_i\|_2 / \|\mathbf{r}_0\|_2 \le Tol$
        то КОНЕЦ ($\mathbf{x}_i$ **– полученное решение**)
увеличить i

2.3.2 Порядок выполнения работы

1. Программно реализовать алгоритм метода сопряженных градиентов.
2. Проверить корректность работы на примере решения СЛАУ ($\mathbf{x}_0$– нулевой вектор, Tol=10^{-3}, N_{it}^{max}=100): **A** = [10 6 2 0; 6 1 5 4; 2 1 1 -2; 0 4 -2 2], **b** = [14; 4; 8; 6] . Использовать произвольное начальное приближение. Построить график зависимости относительной нормы невязки ($\|\mathbf{r}_i\|_2/\|\mathbf{r}_0\|_2$) от номера итерации.
3. Полученные аналогичные результаты с помощью стандартных средств Octave (Scilab).
4. Решить СЛАУ с матрицей Гильберта (матрица **H** с элементами $h_{ij} = 1/(i + j - 1)$, $i, j = 1,\ldots, N$) при $\mathbf{x}_0$– нулевой вектор, Tol=10^{-3}, N_{it}^{max}=100) для порядков матрицы $N = 10, 11, \ldots, 14$. Для задания вектора свободных членов **b** воспользоваться заранее заданным вектором решения **x** в виде 1, 2, …, N. Результаты вычислений (вектор решения) занести в таблицу.
5. Решить те же СЛАУ с помощью стандартного средства Octave или Scilab (x=A\b). Результаты вычислений (вектор решения) занести в таблицу.
6. При решении методом сопряженных градиентов, изменяя точность вычислений (Tol=10^{-4}, 10^{-6}, 10^{-8}) получить вектор решения при порядке матрицы СЛАУ $N = 15$. Полученные значения занести в таблицу с указанием числа требуемых итераций. Для каждого случая построить график зависимости относительной нормы невязки ($\|\mathbf{r}_i\|_2/\|\mathbf{r}_0\|_2$) от номера итерации.
7. Проанализировать полученные результаты и сформулировать выводы по работе.
8. Оформить отчет.

2.3.3 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического

профиля. Общие требования и правила оформления".

2.4 Предобусловливание

Цель работы – получение навыков использования предобусловливания при итерационном решении СЛАУ.

Для достижения поставленной цели используется программное обеспечение GNU Octave.

2.4.1 Краткая теоретическая справка

Вычисление итерационными методами зависит от обусловленности матрицы **A**, оцениваемой числом обусловленности $\mathrm{cond}\mathbf{A} = \|\mathbf{A}\|\cdot\|\mathbf{A}^{-1}\| \approx$ $\approx |\lambda_{max}|/|\lambda_{min}|$. С ростом $\mathrm{cond}\mathbf{A}$ обусловленность ухудшается, и для ряда проблем сходимость может оказаться очень медленной, и поэтому итерационный процесс может застопорится (стагнировать) или даже оборваться. Однако применение, так называемого, предобусловливания улучшает сходимость к требуемому решению.

Пусть **M** – некоторая невырожденная $N\times N$ матрица. Помножив **A** на матрицу $\mathbf{M}^{-1}$, получим систему

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}, \qquad (2.10)$$

которая в силу невырожденности **M** имеет то же точное решение $\mathbf{x}_*$. После обозначений $\widehat{\mathbf{A}} = \mathbf{M}^{-1}\mathbf{A}$ и $\widehat{\mathbf{b}} = \mathbf{M}^{-1}\mathbf{b}$ (2.10) примет вид

$$\widehat{\mathbf{A}}\,\mathbf{x} = \widehat{\mathbf{b}}\,. \qquad (2.11)$$

Хотя (2.11) алгебраически эквивалентна уравнению $\mathbf{A}\mathbf{x} = \mathbf{b}$, спектральные характеристики матрицы $\widehat{\mathbf{A}}$ отличаются от характеристик матрицы **A**, что ведет к изменению скорости сходимости итерационных методов для (2.11) по отношению к (2.11) в конечной арифметике.

Процесс перехода от $\mathbf{A}\mathbf{x} = \mathbf{b}$ к (2.11) с целью улучшения характеристик матрицы для ускорения сходимости к решению называется **предобусловливанием**, а матрица $\mathbf{M}^{-1}$ – матрицей предобусловливателя. Из (2.10) сразу же вытекает важное требование: матрица **M** должна быть близка к матрице **A**. Выбор $\mathbf{M} = \mathbf{A}$ сразу же приводит исходное уравнение $\mathbf{A}\mathbf{x} = \mathbf{b}$ к виду

$\mathbf{Ix} = \mathbf{A}^{-1}\mathbf{b}$, однако не имеет практического смысла, так как требует нахождения $\mathbf{A}^{-1}$, что, по существу, и сводится к решению исходного уравнения. Вторым естественным требованием является требование легкой вычислимости матрицы **M**.

Невязкой системы, соответствующей вектору **x**, называется вектор $\mathbf{r} = \mathbf{b} - \mathbf{Ax} = \mathbf{A}(\mathbf{x}_* - \mathbf{x})$, где $\mathbf{x}_*$ – точное решение. Невязка $\bar{\mathbf{r}}$ системы (2.11) связана с невязкой системы $\mathbf{Ax} = b$ очевидным соотношением

$$\mathbf{M}\bar{\mathbf{r}} = \mathbf{r}, \tag{2.12}$$

которое справедливо и для произведений вида $\mathbf{z} = \mathbf{Aq}$ и $\bar{\mathbf{z}} = \hat{\mathbf{A}}\mathbf{q}$:

$$\bar{\mathbf{z}} = \hat{\mathbf{A}}\mathbf{q} = \mathbf{M}^{-1}\mathbf{Aq} \Rightarrow \mathbf{M}\bar{\mathbf{z}} = \mathbf{Aq} = \mathbf{z}. \tag{2.13}$$

Это позволяет вместо явного перехода от $Ax = b$ к (2.11) вводить в схемы методов корректирующие шаги для учета влияния предобусловливающей матрицы (см. алгоритм метода BiCGstab). В некоторых случаях, когда матрица **M** имеет простую структуру, переход от **A** и **b** к $\hat{\mathbf{A}}$ и $\hat{\mathbf{b}}$ может выполняться и явно.

Из (2.13) следует еще одно условие: структура матрицы предобусловливателя должна допускать легкое и быстрое решение «обратных к предобусловливателю» систем вида $\mathbf{M}\bar{\mathbf{z}} = \mathbf{z}$.

Таким образом, матрица **M** должна быть:

- близка к матрице **A**;
- легко вычислима;
- легко обратима.

Описанное выше предобусловливание иногда называют левым, так как умножение матрицы СЛАУ на матрицу предобусловливателя выполняется слева. Далее в работе используется такой вид предобусловливания на примере итерационного метода BiCGStab, предназначенного для решения СЛАУ с несимметричной матрицей. Данный метод также является представителем класса проекционных методов Крыловского типа, как и метод сопряженных градиентом, рассмотренный на предыдущем занятии. Далее приведен алгоритм метода BiCGStab.

Алгоритм метода BiCGStab

Если необходимо построить матрицу предобусловливателя M

Выбрать начальное приближение $\mathbf{x}^{(0)}$

$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\,\mathbf{x}^{(0)}$

Выбрать вектор $\tilde{\mathbf{r}}$**, удовлетворяющий условию** $(\mathbf{r}^{(0)}, \tilde{\mathbf{r}}) \neq 0$ (например, $\tilde{\mathbf{r}} = \mathbf{r}^{(0)}$)

Для $i = 1, 2, \ldots$ **до сходимости или до** N_{it}^{max}

$\rho_{i-1} = (\tilde{\mathbf{r}}, \mathbf{r}^{(i-1)})$

Если $\rho_{i-1} = 0$

то метод не может решить данную систему

Если $i = 1$

$\mathbf{p}^{(i)} = \mathbf{r}^{(i-1)}$

Иначе

$\beta_{i-1} = (\rho_{i-1} / \rho_{i-2})\,(\alpha_{i-1} / \omega_{i-1})$

$\mathbf{p}^{(i)} = \mathbf{r}^{(i-1)} + \beta_{i-1}(\mathbf{p}^{(i-1)} - \omega_{i-1}\,\mathbf{v}^{(i-1)})$

Найти $\hat{\mathbf{p}}$ **из системы** $\mathbf{M}\hat{\mathbf{p}} = \mathbf{p}^{(i)}$

$\mathbf{v}^{(i)} = \mathbf{A}\hat{\mathbf{p}}$

$\alpha_i = \rho_{i-1} / (\tilde{\mathbf{r}}, \mathbf{v}^{(i)})$

$\mathbf{s} = \mathbf{r}^{(i-1)} - \alpha_i\,\mathbf{v}^{(i)}$

Если $\|\mathbf{s}\|_2 / \|\mathbf{r}^{(0)}\|_2 \leq Tol$

то КОНЕЦ ($x^{(i)} = x^{(i-1)} + \alpha_i\hat{\mathbf{p}}$ – полученное решение)

Найти $\hat{\mathbf{s}}$ **из системы** $\mathbf{M}\hat{\mathbf{s}} = \mathbf{s}$

$\mathbf{t} = \mathbf{A}\hat{\mathbf{s}}$

$\omega_i = (\mathbf{t}, \mathbf{s}) / (\mathbf{t}, \mathbf{t})$

$\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i\hat{\mathbf{p}} + \omega_i\hat{\mathbf{s}}$

$\mathbf{r}^{(i)} = \mathbf{s} - \omega_i\,\mathbf{t}$

Если $\|\mathbf{r}\|_2 / \|\mathbf{r}^{(0)}\|_2 \leq Tol$

то КОНЕЦ ($x^{(i)}$ – полученное решение)

увеличить i

Рассмотрим синтаксические особенности метода BiCGStab при использовании стандартных средств Octave. Синтаксис достаточно гибкий, но имеет ряд особенностей. Так, возможные варианты использования следующие:

x = bicgstab(A,b)
bicgstab(A,b,tol)
bicgstab(A,b,tol,maxit)

bicgstab(A,b,tol,maxit,M)
bicgstab(A,b,tol,maxit,M1,M2)
bicgstab(A,b,tol,maxit,M1,M2,x0)
[x,flag] = bicgstab(A,b,...)
[x,flag,relres] = bicgstab(A,b,...)
[x,flag,relres,iter] = bicgstab(A,b,...)
[x,flag,relres,iter,resvec] = bicgstab(A,b,...)
...

Также можно воспользоваться «пустым» параметром ([]), например:

x = bicgstab (A, b, tol, maxit, M1, [], x0)
x = bicgstab (A, b, tol, [], M1, [], x0)

Рассмотрим используемые входные параметры.

Если метод не сходится при требуемой точности за максимальное число итераций, то выводится сообщение, включающее информацию о величине достигнутой относительной нормы невязки (Bicgstab stopped at iteration 15 without converging to the desired tolerance 1.000000e-012 because the maximum number of iterations was reached. The iterate returned (number 15) has relative residual 9.475671e-008).

Точность вычислений задается с помощью параметра *tol*. Если величина не указана, то она принимается по умолчанию равной 10^{-6}.

Максимальное число итераций задается параметром *maxit*. Если он пропущен или задан «пустым», то используется значение по умолчанию, равное 20.

Если метод расходится, то выдается соответствующее сообщение, содержащее информацию о достигнутой величине относительной нормы невязки (Bicgstab stopped at iteration 16 without converging to the desired tolerance 1.000000e-012 because the method stagnated. The iterate returned (number 15) has relative residual 9.475671e-008).

Параметры *M1* и *M2* предназначены для задания предобусловливателя в виде $\mathbf{M} = \mathbf{M}_1 * \mathbf{M}_2$.

Параметр *x0* определяет начальное приближение, по умолчанию это нулевой вектор.

Далее рассмотрим назначение выходных параметров:

- *flag* – флаг сходимости (0 – метод сошелся с требуемой точностью; 1 – метод не сошелся за максимальное число итераций; 2 – предобусловливатель плохо обусловлен; 3 – метод стагнировал (на соседних итерациях получены схожие результаты; 4 – значение одной из скалярных величин (α, ω, и др.), используемых при реализации метода, слишком велико или слишком мало для продолжения вычислений);
- *relres* – достигнутое значение евклидовой нормы невязки, относительно её начального значения;
- *iter* – число итераций, потребовавшееся для получения решения;
- *resvec* – вектор, содержащий значения евклидовой нормы невязки, полученное в конце каждой итерации (**данный вектор содержит «историю сходимости» метода, т.е. в *resvec*(*i*) хранится евклидова норма невязки после (*i*–1)-й итерации**).

В качестве примера рассмотрим следующую программу.

```
A = gallery ("dorr", 10);
[n,n]=size(A)
x=1:n;
b=A*x';
tol = 1e-12;
maxit = 10;
 [x1,flag1,relres1,iter1,resvec1] = bicgstab(A,b,tol,maxit);
semilogy(1:iter1-1,resvec1/norm(b),'-o');
xlabel('Iteration number');
ylabel('Relative residual');
flag1
relres1
iter1
```

В результате выполнения, получим:

flag1 = 1

relres1 = 3.2206e-012

iter1 = 10

Таким образом, метод не сошелся за требуемое число итераций. На рисунке 2.3 приведена зависимость относительной нормы невязки от номера итерации

(«история сходимости»).

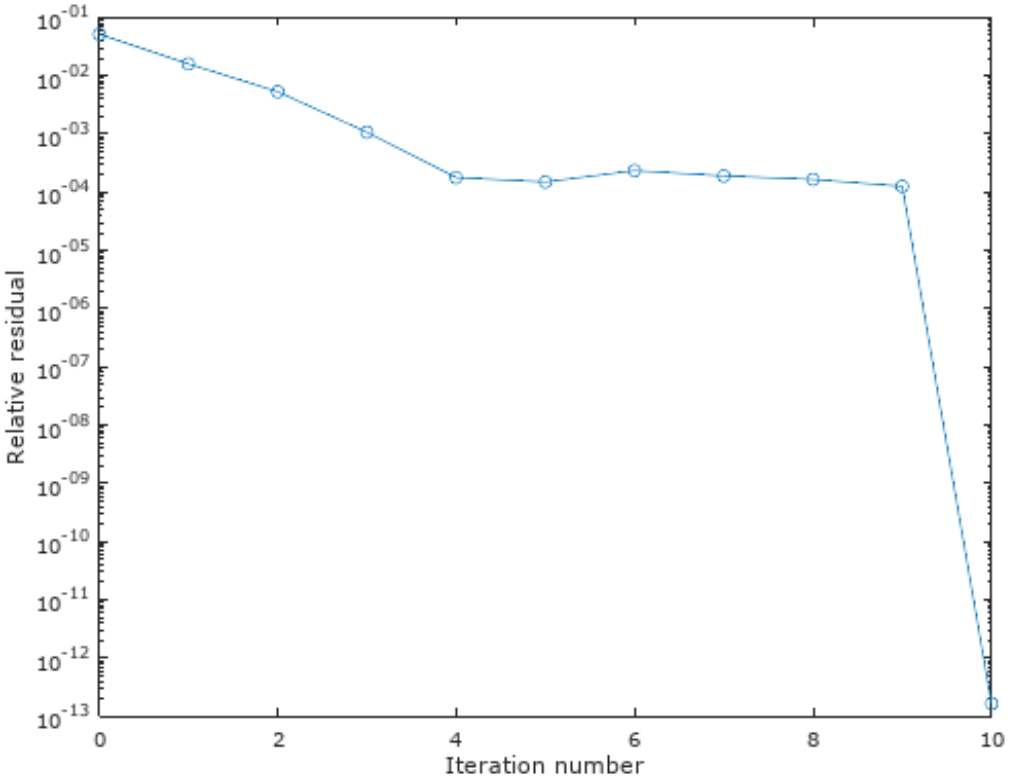


Рисунок 2.3 – Зависимость относительной нормы невязки метода BiCGStab от номера итерации

Для «улучшения» сходимости, как было отмечено выше, используется предобусловливание. Данная работа посвящена изучению двух способов формирования матрицы **M**: диагонального и на основе неполного LU-разложения. В первом случае матрица **M** состоит из одной главной диагонали матрицы **A** (M = diag(diag(A))). Во втором случае имеет место равенство

$$\mathbf{M} = \mathbf{LU} + \mathbf{R}, \qquad (2.14)$$

где **L** и **U** – нижне- и верхнетреугольные матрицы, а матрица **R** является матрицей ошибки. Тогда приближенное представление $\mathbf{M} \approx \mathbf{LU}$ называется неполным LU-разложением матрицы **M** или коротко, её ILU-разложением. Самой простой версией данного разложения является ILU(0). Оно заключается в применении LU-разложения к матрице **A**, но если $a_{ij} = 0$, то сразу полагается $l_{ij} = 0$ или $u_{ij} = 0$. Если $NZ(\mathbf{A})$ – структура разреженности матрицы **A**, а $NZ(\mathbf{M})$ – матрицы **M**, то очевидно, что данный способ приводит к тому, что их структуры совпадают. (Чтобы подчеркнуть тот факт, что в данной постановке

задачи в структуру не вводятся новые ненулевые элементы, такую факторизацию часто называют факторизацией с нулевым заполнением.) Это удобно, когда исходная и разреженная матрицы хранятся с помощью специальных форматов хранения, учитывающих только ненулевые элементы.

Поскольку полное разложение приводит к заполнению структуры матрицы, без каких либо ограничений, а ILU(0)-разложение – к нулевому заполнению, очевидно, что еще одним вариантом может служить разложение с неким уровнем заполнения. Одной из таких стратегий расширения структуры разреженности является использование постфильтрации, позволяющей контролировать уплотнение структуры. Данное разложение называется неполным LU-разложением с упороживанием и выбором ведущего элемента или ILUTP-разложение. Оба вида разложения (а также некоторые другие) могут быть использованы в Octave в виде

ilu(A,setup)
[L,U] = ilu(A,setup)

Параметр *setup* – структура данных, имеющая несколько полей, используемых для настроек вычислений. Можно активировать любое количество этих полей и определить их в любой последовательности. В данной работе используются следующие поля:

- *type* – тип разложения. По умолчанию используется '*nofill*'. При этому будет реализовано ILU(0)-разложение. В случае реализации ILUTP-разложения данное поле принимает значение '*ilutp*'.
- droptol – порог обнуления (неотрицательное число). По умолчанию значение равно нулю, что соответствует полному LU-разложению. При этом элементы треугольной матрицы **U** обнуляются (кроме диагональных элементов), если *abs(U(i,j)) < droptol*norm(A(:,j))*, а матрицы **L**, если *abs(L(i,j)) < droptol*norm(A(:,j))/U(j,j)*.

Далее приведен пример использования описанного функционала.

```
setup.type = 'ilutp';
setup.droptol = 0.01;
```

[L,U] = ilu(A,setup);

2.4.2 Порядок выполнения работы

1. Изучить алгоритм метода BiCGStab.
2. Написать программу для решения СЛАУ с матрицей, формируемой с помощью команды: A = gallery ('neumann', 1600) + speye (1600).
3. Определить порядок матрицы *N*, число ненулевых элементов, число обусловленности и получить её портрет. Значения сохранить.
4. Вектор **b** определить путем первоначального задания вектора решения **x** значениями от 1 до *N*.
5. Проанализировать сходимость метода без использования предобусловливания при нулевом начальном приближении ($\mathbf{x}_0 = 0$), требуемой точности 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12} и максимальном числе итераций 15 и 50.
6. Проанализировать сходимость метода с применением диагонального и основанного на ILU(0)-разложении предобусловливания при нулевом начальном приближении, точности 10^{-3} и 10^{-12} и максимальном числе итераций 15. При использовании ILU(0)-разложения получить портрет матрицы **M** и число её ненулевых элементов.
7. Повторить п. 6 с применением ILUTP-разложения при значении параметра *droptol* – 0,1; 0,01 и 0,001. Получить портреты полученных матриц M и число их ненулевых элементов.
8. Для всех вычислений оценить время выполнения, значения свести в таблицу. Оценить время решения данной СЛАУ с помощью следующий команды x=A\b.
9. Для результатов, полученных согласно п. 5–7, построить зависимости относительной нормы невязки метода BiCGStab от номера итерации.
10. Проанализировать полученные результаты и сформулировать выводы по работе.
11. Оформить отчет.

2.4.3 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

2.5 Метод конечных разностей

Цель работы – получение навыков использования и программной реализации метода конечных разностей.

Для достижения поставленной цели используется программное обеспечение GNU Octave или Scilab. В ходе работы необходимо программно реализовать вычисление погонной емкости коаксиальной линии передачи.

2.5.1 Краткая теоретическая справка

Метод конечных разностей (FDM, МКР) впервые был разработан А. Томом в 1920 г. под названием «Метод квадратов» для решения нелинейных уравнений гидродинамики. С тех пор метод нашел применение в решении проблем различных областей. Методы конечных разностей основаны на аппроксимациях, которые позволяют заменять дифференциальные уравнения уравнениями конечных разностей. Эти разностные уравнения имеют алгебраический вид. Они ставят в соответствие значению зависимой переменной в точке расчетной области значение в некоторой соседней точке. Таким образом, решение конечных разностей в основном состоит из трех этапов:

1. разделение области решения на сетку узлов;
2. аппроксимация данного дифференциального уравнения разностным эквивалентом;
3. решение разностных уравнений с учетом заданных граничных

условий и / или начальных условий

В данной работе МКР используется для анализа коаксиального кабеля, поперечное сечение которого приведено на рисунке 2.5. Волновое сопротивление линии передачи определяется с помощью выражения

$$Z_0 = \sqrt{\frac{L}{C}} = \frac{\sqrt{\mu\varepsilon}}{C}, \tag{2.3}$$

где L – погонная индуктивность; C –погонная емкость; μ – абсолютная магнитная проницаемость среды; ε – абсолютная диэлектрическая проницаемость среды; c – скорость света в вакууме. При этом фазовая скорость равна

$$u = \frac{1}{\sqrt{LC}}. \tag{2.4}$$

Таким образом, если считать материалы линии передачи немагнитными, т.е. абсолютная магнитная проницаемость равна магнитной постоянной, то для определения волнового сопротивления линии достаточно знать ее погонную емкость и абсолютную диэлектрическую проницаемость среды.

При удалении из рассматриваемой структуры диэлектрика (замена его на воздух) волновое сопротивление и фазовую скорость можно вычислить с помощью следующих выражений

$$Z_{00} = \sqrt{\frac{L}{C_0}}, \tag{2.5}$$

$$u_0 = \frac{1}{\sqrt{LC_0}} \tag{2.6}$$

где C_0 – погонная емкость структуры без диэлектрика, а $u_0 = c$ – скорость света в свободном пространстве. Комбинируя выражения (2.3)–(2.6), получим

$$Z_0 = \frac{1}{u_0\sqrt{CC_0}} = \frac{1}{uC}, \tag{2.7}$$

$$u = u_0\sqrt{\frac{C_0}{C}} = \frac{u_0}{\sqrt{\varepsilon_{eff}}}, \tag{2.8}$$

$$\varepsilon_{eff} = \frac{C}{C_0}, \tag{2.9}$$

где ε_{eff} – эффективная диэлектрическая проницаемость.

Таким образом, для нахождения волнового сопротивления и фазовой скорости для рассматриваемой структуры с неоднородным диэлектрическим заполнением необходимо нахождение погонной емкости с и без диэлектрической подложки.

Перейдем к определению погонной емкости. Так, в общем случае емкость связана с величиной полного заряда в анализируемой структуре (системе) следующим равенством

$$C = k\frac{q_\Sigma}{\Phi_0}, \tag{2.10}$$

где Φ_0 – потенциал между внутренним (центральным) и внешним проводниками. Коэффициент k – коэффициент, необходимый для учета симметрии. Так, в рассматриваемом случае он равен 4 (полная симметрия). Для нахождения полного заряда воспользуемся законом Гаусса для контура l, охватывающего внутренний проводник. Выберем прямоугольный контур между двумя смежными прямоугольными границами, как показано на рисунке 2.4.

$$\begin{aligned} q_\Sigma = \oint_l \mathbf{D}\cdot d\mathbf{l} = \oint_l \varepsilon\frac{\partial\Phi}{\partial n}dl = \varepsilon\left(\frac{\Phi_P - \Phi_N}{\Delta x}\right)\Delta y + \varepsilon\left(\frac{\Phi_M - \Phi_L}{\Delta x}\right)\Delta y + \\ + \varepsilon\left(\frac{\Phi_H - \Phi_L}{\Delta y}\right)\Delta x + \varepsilon\left(\frac{\Phi_G - \Phi_K}{\Delta y}\right)\Delta x + \ldots \end{aligned}, \tag{2.11}$$

Полагая $\Delta x = \Delta y = h$, получим

$$q_\Sigma = (\varepsilon\Phi_P + \varepsilon\Phi_M + \varepsilon\Phi_H + \varepsilon\Phi_G + \ldots) - (\varepsilon\Phi_N + 2\varepsilon\Phi_L + \varepsilon\Phi_K + \ldots)$$

или

$q_\Sigma = \varepsilon_0 \sum \varepsilon_{ij} \Phi_i -$

(для i – х узлов, расположенных на внешней границе $GHJMP$,

без учета углов, таких как J)

$- \varepsilon_0 \sum \varepsilon_{ij} \Phi_i$

(для i – х узлов, расположнных на внутреней границе KLN

с двойным учетом углов, таких как L),

где Φ_i и ε_{ri} – потенциал и диэлектрическая проницаемость i-го узла соответственно. Если i-й узел расположен на границе диэлектрик-диэлектрик, то $\varepsilon_{ri}=(\varepsilon_{r1}+\varepsilon_{r2})/2$. Если же он расположен на оси симметрии, то $\Phi_i = \Phi_i/2$ (поскольку симметрия учтена в уравнении (2.10)).

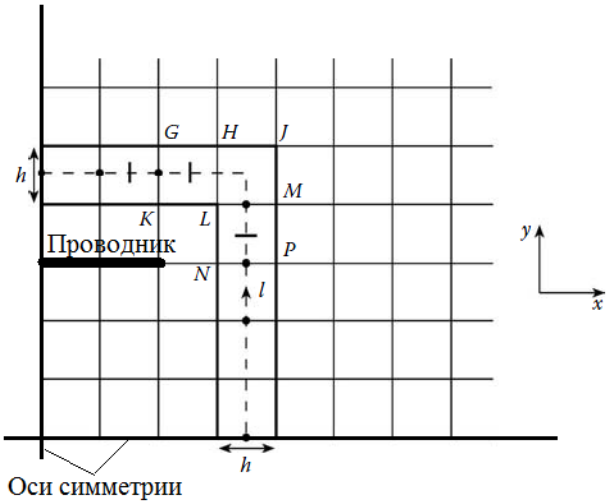


Рисунок 2.4 – Прямоугольный контур l, используемый для нахождения сосредоточенного заряда на поверхности проводника

Далее необходимо аналогично вычислить C_0, удалив диэлектрические границы (положив для всех узлов $\varepsilon_{ri}=1$). После чего можно вычислить волновое сопротивление по формуле (2.7).

Отдельно стоит отметить, что при вычислениях часто проверяют не сходимость первичного параметра (потенциала) в узлах (см. выше), а вторичного (погонная емкость), для чего выполняется проверка

$$error = \max\left(\left|\frac{C_{ij}^{(it)} - C_{ij}^{(it-1)}}{C_{ij}^{(it)}}\right|\right) \le tol.$$

2.5.2 Порядок выполнения работы

1. Программно реализовать вычисление погонной емкости МКР коаксиального кабеля при $a = b = 1$, $c = d = 2$ (усл. ед.), потенциал центрального проводника 1 В, экрана 0 В, относительная диэлектрическая проницаемость равна 1:

a. Без учета симметрии.

b. С учетом симметрии.

2. Вычисления должны сопровождаться оценкой вычислительных затрат (число итераций, время решения, затраты памяти рабочей станции).

3. При вычислениях для останова процесса итерационного процесса использовать относительную погрешность погонной емкости: 10^{-9}.

4. Без учета симметрии вычисления должны быть выполнены при делении рассчитываемой структуры на количество подынтервалов: 10; 20; 30; 40; 50 (при этом должна быть выполнена оценка шага сетки h).

5. С учетом симметрии вычисления должны быть выполнены при делении рассчитываемой структуры на количество подынтервалов: 5; 10; 15; 20; 25 (при этом должна быть выполнена оценка шага сетки h).

6. Полученные результаты свести в соответствующие таблицы.

7. Проанализировать полученные результаты и сформулировать выводы по работе.

8. Оформить отчет.

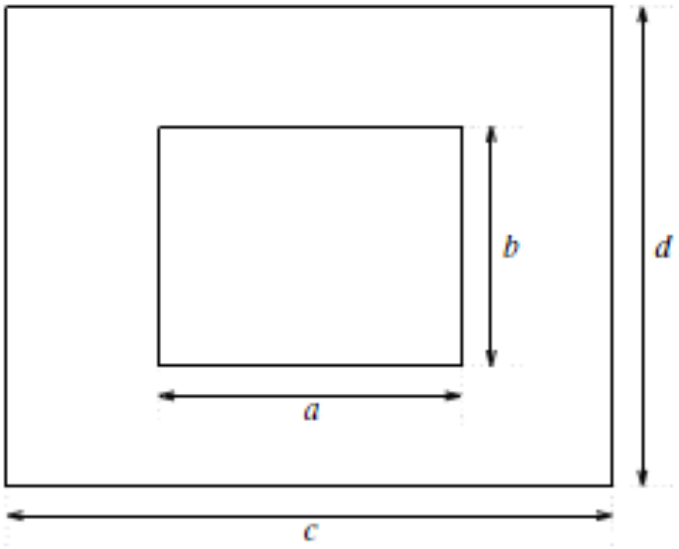


Рисунок 2.5 – Поперечное сечение коаксиального кабеля

2.5.3 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

2.6 Метод моментов: базисные функции

Цель работы – получение навыков задания, использования и программной реализации базисных функций метода моментов.

Для достижения поставленной цели используется программное обеспечение GNU Octave или Scilab. В ходе работы необходимо программно реализовать три вида базисных функций и с их помощью аппроксимировать заданную функцию.

2.6.1 Краткая теоретическая справка

Рассмотрим детерминированное уравнение:

$$Lf = g, \tag{2.12}$$

где L – линейный оператор (дифференциальный, интегральный или интегро-дифференциальный), g – известная функция, а f – искомая неизвестная функция. Пусть f представляется системой базисных функций $\{f_1, f_2, f_3, \dots\}$ в области L как линейная комбинация

$$f = \sum_{j=1}^{N} a_j f_j, \tag{2.13}$$

где a_j – искомые скаляры. Для приближённых решений (2.13) является конечной суммой, а для точных – обычно бесконечной. Подставляя (2.13) в (2.12) и используя линейность L, получим

$$\sum_j a_j L f_j = g, \tag{2.14}$$

где равенство является обычно приближённым.

Выбрав соответствующее скалярное произведение, и определив систему тестовых (весовых) функций $\{w_1, w_2, w_3, \dots\}$ в области определения L, получим (после взятия скалярного произведения от обеих частей уравнения (2.14)

$$\sum_j \alpha_j < w_i, L f_j > = < w_i, g >, \; i = 1, 2, 3, \dots . \tag{2.15}$$

Эту систему уравнений можно записать в матричном виде как

$$[l]\underline{\alpha} = \underline{g}, \tag{2.16}$$

где $[l]$ является матрицей

$$[l] = [< w_i, L f_j >], \tag{2.17}$$

а $\underline{\alpha}$ и $\underline{g}$ являются векторами-столбцами

$$\underline{\alpha} = [\alpha_j]; \tag{2.18}$$

$$\underline{g} = [< w_i, g >]. \tag{2.19}$$

Если [l] несингулярна, то существует её инверсия, и $\underline{\alpha}$ находится как

$$\underline{\alpha} = [l]^{-1} \underline{g}. \tag{2.20}$$

Решение для f теперь даётся посредством (2.13). Для краткого обозначения зададим вектор-строку функций

$$\tilde{f} = [f_j]. \tag{2.21}$$

Записав (2.13) как $f = \tilde{f}\underline{\alpha}$ и подставив $\underline{\alpha}$ из (2.20), получим решение

$$f = \tilde{f}[l]^{-1} \underline{g}, \tag{2.22}$$

которое может быть приближённым или точным в зависимости от выбора функций разложения и тестовых функций.

Для любого численного метода важно выяснить вопросы, связанные с его сходимостью, скоростью сходимости и точностью, которые для метода моментов зависят от оператора $\boldsymbol{L}$, базисных функций f_i, весовых функций w_i (i=1,2,…N) и их числа N.

Эффективность метода для получения результата с заданной точностью определяется, в конечном счёте, затратами времени и памяти компьютера. Если пока не рассматривать задачу синтеза, а ограничиться задачей одновариантного анализа, то для метода моментов решение сводится к следующим шагам:

- получение из уравнений Максвелла интегральных уравнений структуры;
- дискретизация структуры (разбиение структуры на N подобластей, в каждой из которых искомая функция аппроксимируется базисными функциями);
- вычисление элементов матрицы систем линейных алгебраических уравнений (СЛАУ) размером N*N;
- вычисление элементов вектора воздействий размером N;
- решение СЛАУ;

- вычисление требуемых характеристик из вектора решения СЛАУ.

Эффективность МоМ во многом зависит от выбора ячеек и набора базисных и пробных (тестовых, весовых) функций. Этот выбор определяется формой всей проводящей поверхности. Если эта поверхность (или совокупность поверхностей) имеет форму многоугольника с прямыми углами, целесообразно использовать ячейки прямоугольной формы с размерами $h_x \times h_y$. Размеры ячейки должны выбираться так, чтобы на минимальной длине волны λ_{min} укладывалось не менее 10 шагов (ячеек). Системы базисных функций делятся на два вида: функции подобластей и функции полной области.

Наиболее простые базисные функции имеют постоянное значение в ячейке и равны нулю вне ее. Такие функции, называемые кусочно-постоянными (КПБФ) или импульсные (pulse functions), имеет вид, показанный на рисунке 2.6. При этом интересующий интервал делится на N подынтервалов/импульсов. На рисунке подынтервалы имеют одинаковую длину, но это условие не является обязательным. Согласно рисунку 2.6 КПБФ определяются следующим образом

$$f_m(x) = \begin{cases} 1, & x_m \le x \le x_{m+1} \\ 0, & \text{иначе} \end{cases}.$$

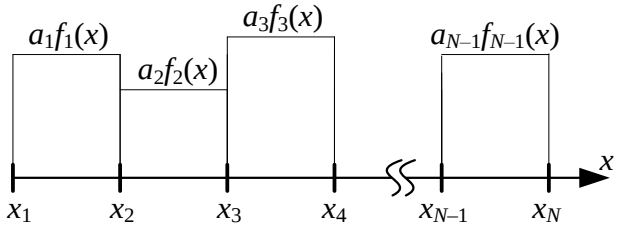


Рисунок 2.6 – Кусочно-постоянные базисные функции

Пример использования КПБФ для аппроксимации на интервале [0, 1.5] функции $f(x) = \dfrac{1}{\sqrt{\left(\dfrac{\pi}{2}\right)^2 - x^2}}$ при N = 10 и 25 приведен на рисунке 2.7. Далее приведен пример программного кода, реализующий данную аппроксимацию.

Недостатком КПБФ является то, что получаемая в результате их

использования аппроксимирующая функция имеет разрывы, что в электродинамике в ряде случаев неприемлемо, поскольку такие функции могут порождать сингулярные поля, не отвечающие физической реальности. Тем не менее, рассматриваемая система базисных функций нашла достаточно широкое применение.

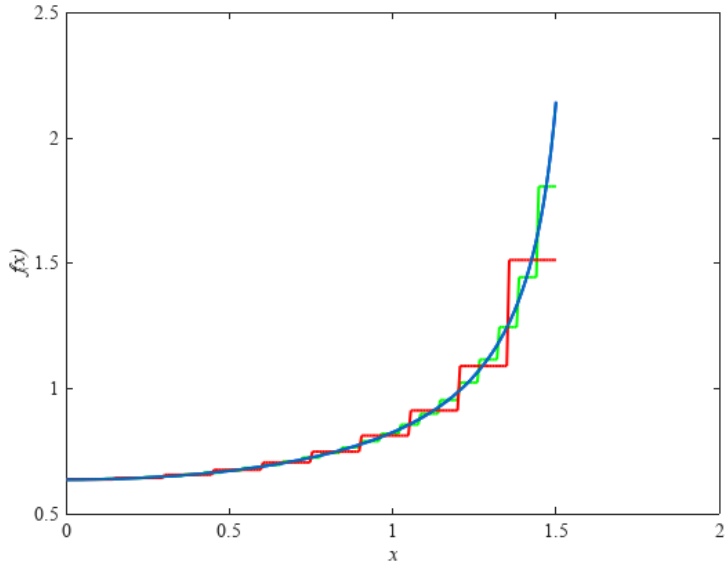


Рисунок 2.7 – Аппроксимация функции $f(x)$ – синяя линия,
с помощью 10 (красная) и 25 (зеленая) КПБФ

Более сложные базисные функции предполагают изменение тока вдоль одной или обеих координат в пределах ячейки. Так, если ток течет в направлении оси x и его можно считать постоянным вдоль оси y, в качестве базисной можно использовать так называемые «крышечные» или кусочно-линейные базисные функции (КЛБФ). Семейство таких функций приведено на рисунке 2.8. Интервал делится на N точек и N–1 подынтервалов, что требует использования N–2 БФ. Длина КЛБФ взята одинаковой, но, как и в случае с импульсными функция, это не обязательное условие. Поскольку смежные функции перекрывают один сегмент, то использование треугольников позволяет кусочно линеаризовать решение между сегментами. КЛБФ определяются следующим образом

$$f_m(x) = \begin{cases} 1 - \dfrac{x_m - x}{h_x}, & x_m - h_x \le x < x_m; \\ 1 + \dfrac{x_m - x}{h_x}, & x_m \le x < x_m + h_x; \\ 0, & \text{иначе.} \end{cases}$$

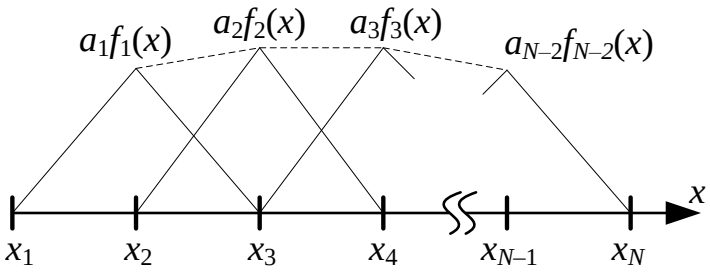


Рисунок 2.8 – Семейство кусочно-линейных базисных функций

Еще раз обратимся к рисунку 2.8. Видно, что при таком использовании решение на концах интервала (x_1 и x_N) равно нулю. Таким образом, такая реализация может быть желательной, когда априорно известно, что значение решения на концах интервала равно нулю априори, но оно не должно использоваться, если решение может быть ненулевым. Если вместо этого добавить первый и последний сегменты полутреугольника, решение больше не будет принудительно сбрасываться в ноль. Это показано на рисунке 2.9. В этом случае используется уже N КЛБФ.

Пример использования КПБФ для аппроксимации функции $f(x) = \dfrac{1}{\sqrt{\left(\dfrac{\pi}{2}\right)^2 - x^2}}$ при N= 10 и 25 приведен на рисунке 2.10. Далее приведен пример программного кода, реализующий данную аппроксимацию.

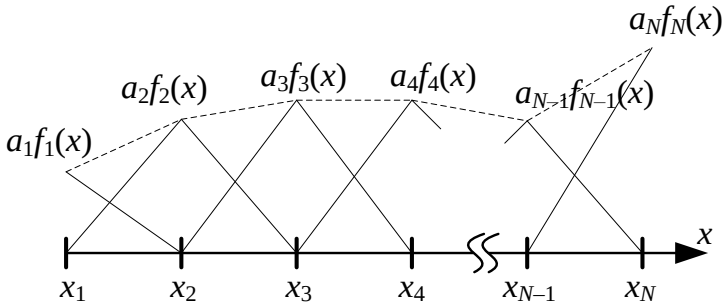


Рисунок 2.9 – Семейство кусочно-линейных базисных функций с дополнительными функциями на концах интервала

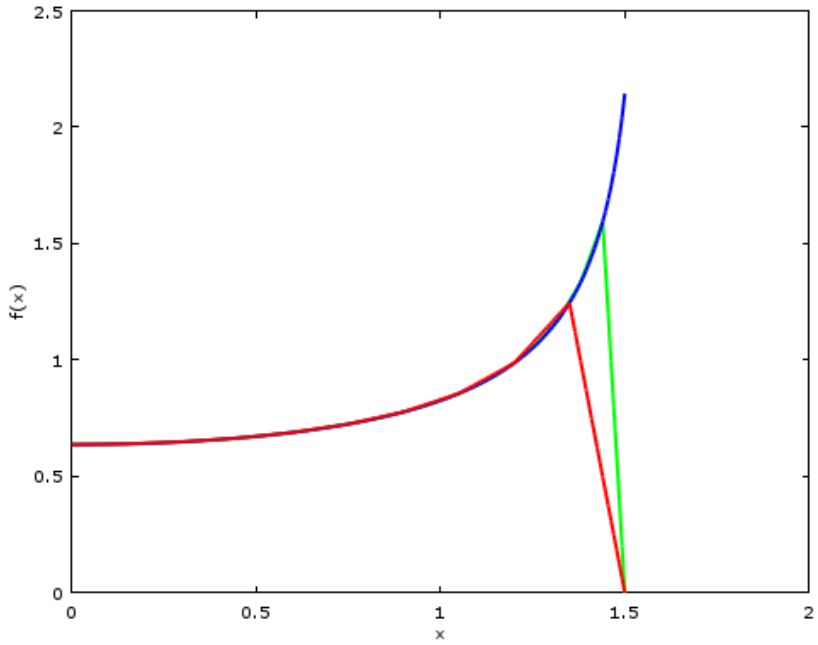


Рисунок 2.10 – Аппроксимация функции $f(x)$ – синяя линия, с помощью 10 (красная) и 25 (зеленая) КЛБФ

Для моделирования быстро изменяющихся в пространстве токов удобнее использовать кусочно-синусоидальные базисные функции (КСБФ), которые подобны КЛБФ, как показано на рисунке 2.11. Они часто используются при анализе проводных антенн из-за их способности представлять синусоидальные распределения токов. Эти функции определяются как

$$f_m(x) = \begin{cases} \dfrac{\sin[k(x-(x_m-h_x))]}{\sin(kh_x)}, & x_m - h_x \le x < x_m; \\ \dfrac{\sin[k(x_m+h_x)-x)]}{\sin(kh_x)}, & x_m \le x < x_m + h_x; \\ 0, & \text{иначе.} \end{cases}$$

где $k = 2\pi/\lambda$ – волновое число. При этом требуется, чтобы длина сегмента была значительно меньше периода используемой синусоиды.

Пример использования КСБФ для аппроксимации функции $f(x) = \dfrac{1}{\sqrt{\left(\frac{\pi}{2}\right)^2 - x^2}}$ приведен на рисунке 2.7. Далее приведен пример программного кода, реализующий данную аппроксимацию.

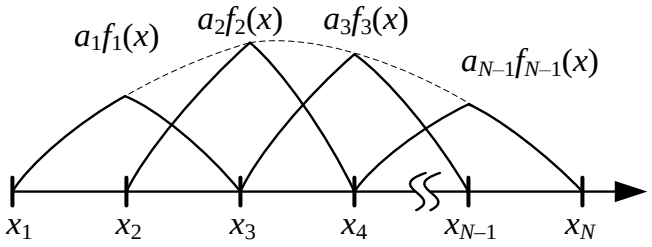


Рисунок 2.11 – Семейство кусочно-синусоидальных базисных функций

2.6.2 Порядок выполнения работы

1. Программно реализовать КЛБФ, КПБФ, КСБФ.
2. Аппроксимировать функцию $y(x) = 1{,}5 - \dfrac{(x-2)^2}{6}$, x = [0;5] при количестве БФ N=3, 5, 10.
3. Выполнить численное сравнение качества аппроксимации.
4. Сформировать выводы по работе.
5. Оформить отчет.

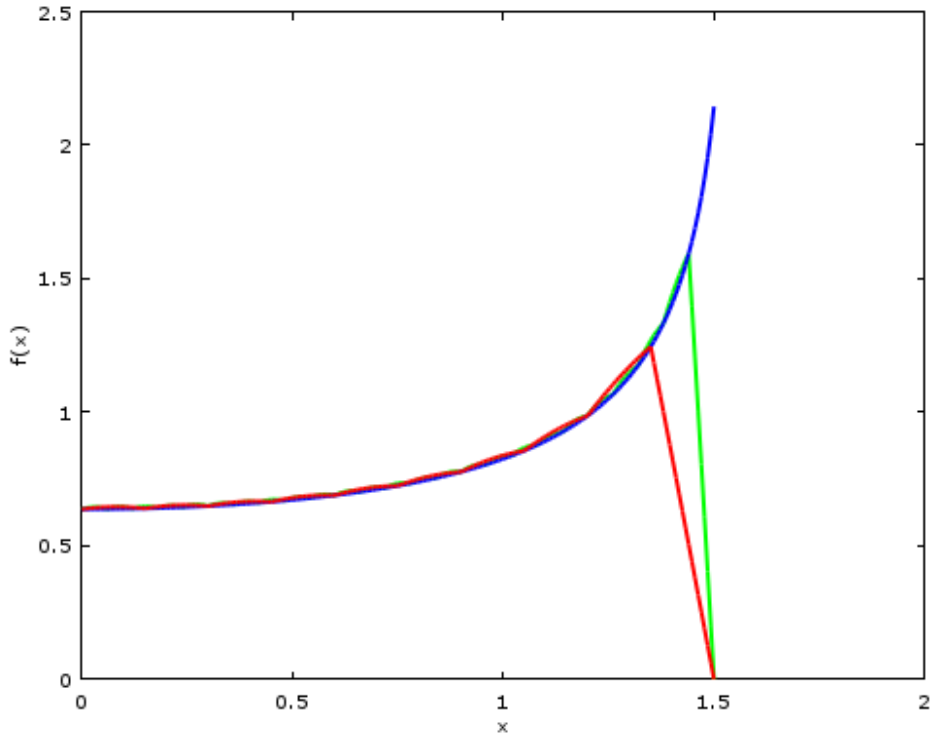


Рисунок 2.12 – Аппроксимация функции $f(x)$ – синяя линия,
с помощью 10 (красная) и 25 (зеленая) КСБФ при $k = 2\pi/20h$

2.6.3 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

2.7 Метод моментов: итерационный выбор сегментации

Цель работы – получение навыков задания, программирования и использования итерационного выбора сегментации анализируемой структуры.

Для достижения поставленной цели анализируется поперечное сечение печатной платы в системе TALGAT. В ходе работы необходимо оценить вычислительные затраты используемых алгоритмов и точность решения.

2.7.1 Краткая теоретическая справка

Моделирование ЭМС отличается особой сложностью, требующей больших вычислительных затрат. Поэтому, для быстрого моделирования необходимо выявлять ресурсы уменьшения затрат на вычисления. Один из них – выбор сегментации структуры. Возможность получения приемлемых результатов даже при самой грубой сегментации делает целесообразным начало моделирования именно с неё, поскольку это требует минимальных затрат. Использование итерационного выбора сегментации представляется эффективным, особенно для моделирования в реальном времени. Рассмотрим один из возможных вариантов его реализации: алгоритм адаптивного итерационного выбора оптимальной сегментации (АИВОС), на примере решения задачи вычисления емкостной матриц (**C**) системы проводников и диэлектриков, порядка N_{COND}, где N_{COND} – количество проводников, не считая опорного. Такая задача сводятся к решению системы линейных алгебраических уравнений (СЛАУ) вида $\mathbf{S\sigma} = \mathbf{V}$, где **S** – квадратная и плотная матрица порядка N, являющаяся результатом применения метода моментов к анализируемой структуре, **V** – матрица, состоящая из задаваемых потенциалов на подобластях, на которые разбиты границы структуры, а $\boldsymbol{\sigma}$ – искомая матрица, дающая распределение поверхностной плотности заряда на этих границах. Порядок матрицы СЛАУ складывается из количества подобластей на границах проводник-диэлектрик (N_C) и диэлектрик-диэлектрик (N_D), а элементы матрицы вычисляются из параметров этих подобластей.

При равномерной сегментации (одинаковой для всей структуры) получим следующий алгоритм вычисления емкостной матрицы.

Алгоритм 1 – исходное вычисление емкостной матрицы системы проводников и диэлектриков

1. Установить исходные параметры моделирования (параметры структуры)
2. Установить равномерную сегментацию (шаг сегментации равен толщине проводника деленное на 3)

3. Вычислить элементы матрицы воздействий **V** и матрицы **S**
4. Найти решение $\boldsymbol{\sigma}$ из уравнения $\mathbf{S}\boldsymbol{\sigma}=\mathbf{V}$

2.7.2 Итерационный выбор сегментации

Далее приведен алгоритм АИВОС.

Алгоритм 2 – Вычисление емкостной матрицы с использованием АИВОС

1. Установить исходные параметры моделирования (*tol*, параметры структуры)
2. Установить длину сегмента равной ширине проводника *w*
3. Вычислить элементы матрицы воздействий $\mathbf{V}^1$ и матрицы $\mathbf{S}^1$
4. Найти матрицу решения $\boldsymbol{\sigma}^1$ из уравнения $\mathbf{S}^1\boldsymbol{\sigma}^1 = \mathbf{v}^1$
5. Вычислить элементы матрицы $\mathbf{C}^1$, основываясь на элементах $\boldsymbol{\sigma}^1$
6. Сохранить значение одного элемента матрицы $\mathbf{C}^1$ (в данной работе будет использована емкость центрального проводника, c_{66}^1.
7. Для i=2, 3, …
8. Уменьшить длину сегмента в два раза
9. Вычислить элементы вектора воздействия $\mathbf{v}^i$ и матрицы $\mathbf{S}^i$
10. Найти вектор решения $\boldsymbol{\sigma}^i$ из уравнения $\mathbf{S}^i\boldsymbol{\sigma}^i = \mathbf{v}^i$
11. Вычислить элементы матрицы $\mathbf{C}^i$, основываясь на элементах $\boldsymbol{\sigma}^i$
12. Сохранить значение одного элемента матрицы $\mathbf{C}^i$ (например c_{66}^i).
13. Если $(c_{66}^i - c_{66}^{i-1})/ c_{66}^i < tol$
14. то Выход, $\mathbf{C}^i$ – требуемое решение
15. Увеличить i

В данной работе также будут исследоваться еще несколько вариантов реализации алгоритма АИВОС:

1. Для останова итераций используется норма Фробениуса емкостной матрицы, вычисляемая по формуле, далее называемый **алгоритм 3**.

$$\|\mathbf{C}\| = \sqrt{\sum_{i,j}^{N_{COND}} c_{i,j}^2} \ ,$$

где $C_{i,j}$ – элемент матрицы **C**, N_{COND} – число проводников системы проводников и диэлектриков не считая опорного.

2. Для останова итераций используется норма Фробениуса матрицы решения СЛАУ (**σ**), далее называемый **алгоритм 4**.

2.7.3 Порядок выполнения работы

Для исследования рассмотренных алгоритмов, в ходе работы используется структура, поперечное сечение которой представлено на рисунке 1 (файл с конфигурацией получить у преподавателя). Геометрические параметры структуры: ширина проводника w=890 мкм, зазоры s_1=500 мкм, s_2=1890 мкм, толщина проводника и сплошных проводящих областей t=35 мкм, толщина препрегов h_1=h_3=144 мкм, толщина подложки h_2=220 мкм. Толщина паяльной маски принята равной h_M=30 мкм. Ширина сплошных проводящих областей принята равной 5w, диэлектрическая проницаемость er_1=er_3=4,5, er_2=5,4.

1. Получить значение емкости центрального проводника c_{66}, время решения (T) и порядок матрицы СЛАУ (N) с использованием алгоритма 1. Шаг сегментации задать равным t/3. Полученные результаты занести в соответствующие строки табл. 1 и 2.
2. Реализовать алгоритм 2. Вычисления выполнять для tol=0,01 и tol=0,001. Получить значение центрального проводника c_{66}, время решения, порядок матрицы СЛАУ и число требуемых итераций (N_{it}). Полученные результаты занести в соответствующие строки табл. 1 и 2.
3. Выполнить вычисления согласно п.2 с использованием алгоритма 3.
4. Выполнить вычисления согласно п.2 с использованием алгоритма 4.
5. Предложить и исследовать свой вариант реализации итерационного выбора сегментации (**алгоритм 5**). Полученные результаты занести в соответствующие строки табл. 1 и 2.

Таблица 2.1 – Результаты вычислений погонной емкости центрального проводника при *tol*=0,01

Результаты	Алгоритм 1	Алгоритм 2	Алгоритм 3	Алгоритм 4	Алгоритм 5
c_{66}, пФ/м					
T, с					
N					
Число итераций					

Таблица 2.2 – Результаты вычислений погонной емкости центрального проводника при *tol*=0,001

Результаты	Алгоритм 1	Алгоритм 2	Алгоритм 3	Алгоритм 4	Алгоритм 5
c_{66}, пФ/м					
T, с					
N					
Число итераций					

2.7.4 Тестовая структура

Для тестирования алгоритмов из данной работы используется структура, представленная на рисунке 2.13.

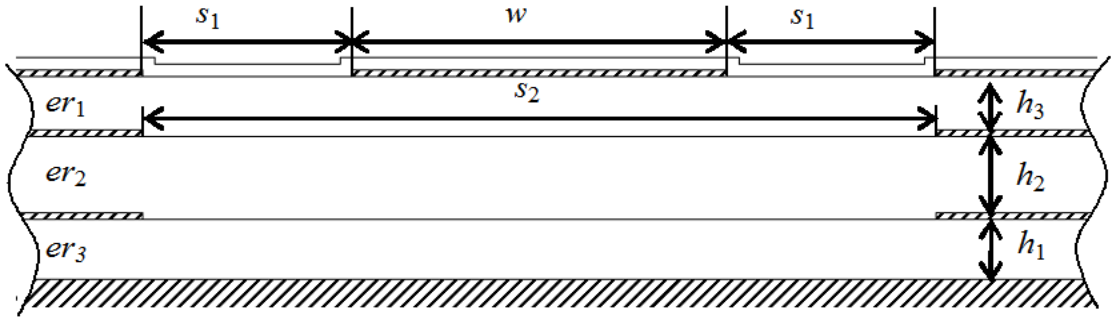


Рисунок 2.13 – Поперечное сечение тестовой структуры

Для экономии времени далее приведен скрипт задания структуры и вычисления матрицы коэффициентов электростатической индукции (емкостной матрицы). После выполнения которого на печать выводится значение c_{66}, соответствующее центральному проводнику, размер матрицы СЛАУ (**S**) и время вычисления.

```
INCLUDE MATRIX
INCLUDE MOM2D
INCLUDE UTIL
SET_AUTO_SEGMENT_LENGTH 35.e-6 // 3 сегмента

SET "t" 35.e-6
SET "w" 890.e-6
SET "h1" 144.6e-6
SET "h2" 220.e-6
SET "h3" 144.6e-6
SET "h_mask" 30.e-6
SET "er1" 4.5
SET "er2" 5.4
SET "er3" 4.5
SET "er_mask" 3.5
SET "s1" 0.5e-3
SET "s2" PLUS w MUL 2.0 s1

CREATE_KEYWORD "etalon"
SET "struct" 3
SET "d1" MUL 5. w
SET "all_w" PLUS PLUS MUL d1 2.0 MUL 2.0 s1 w
SET "d2" DIV MINUS all_w s2 2.0

LAYER h1 er1 0.
IF EQU struct 3
THEN COND 1.e-6 d2 t
THEN COND s2 d2 t
THEN SET "struct" 2

LAYER h2 er2 0.
IF EQU struct 2
THEN COND 1.e-6 d2 t
THEN COND s2 d2 t
THEN SET "struct" 1

LAYER h3 er3 0.
IF EQU struct 0
THEN COND PLUS PLUS 1.e-6 d1 s1 w t

IF EQU struct 1
THEN COND 1.e-6 d1 t
THEN COND s1 w t
THEN COND s1 d1 t

COVER h_mask er_mask 0.

SET "conf" GET_CONFIGURATION_2D
DRAW_CONFIGURATION2D conf

SET "mC" CALCULATE_C SMN_C conf conf
```

```
IF EQU struct 0
THEN SET "C" GET_MATRIX_VALUE mC 0 0
ELSE SET "C" GET_MATRIX_VALUE mC MINUS GET_ROWS mC 2 MINUS GET_ROWS
mC 2

ECHO C
ECHO GET_SIZE SMN_C conf

END_CREATE_KEYWORD

REPORT_TIMER etalon
```

2.7.5 Методические указания

В работе рекомендуется использовать следующий инструментарий: кейворды (аналог функций из языков программирования), цикл for, команду вычисления нормы матрицы, команду вычисления матрицы (вектора) решения и оператор if. Далее приведен пример создания кейворда с именем Calc (следует обратить внимание, что имя кейворда из данного примера отличен от имени кейворда, приведенного в предыдущем подразделе)

```
CREATE_KEYWORD "Calc"
        CONDUCTOR
        LINE 0.5 0.5 1.5 0.5
        LINETO 1.5 1.5
        LINETO 0.5 1.5
        LINETO 0.5 0.5
        CONDUCTOR_GROUNDED
        LINE 0. 0. 2. 0.
        LINETO 2. 2.
        LINETO 0. 2.
        LINETO 0. 0.
END_CREATE_KEYWORD
```

Для его вызова достаточно в требуемое место скрипта поместить Calc (кейвод должен быть определен раньше места вызова). Стоит отметить, что вызов кейводов (функций) может быть организован из тела другого кейвода (функции). Далее приведен пример вызова кейворда (функции) Calc из тела кейворда (функции) adaptive_calc

```
CREATE_KEYWORD "adaptive_calc"
SET "segm" DIV segm 2.0
Calc
END_CREATE_KEYWORD
```

Для создания цикла for переменной *i*, изменяющейся от 1 до 10 с шагом 2 достаточно написать следующею строку скрипта:

```
FOR "i" 1 10 2 Calc
```

который при данных параметрах 5 раз вызовет кейворд (функцию) Calc.

Далее приведена команда, позволяющая вычислить норму некой матрицы **S** и поместить полученное значение в переменную *nS*

```
SET "nS" MATRIX_NORM2 S
```

Напомним, что для вычисления емкостной матрицы структуры используется две команды (conf – переменная, содержащая информацию о конфигурации моделируемой структуры)

```
SET "smnC" SMN_C conf
SET "mC" CALCULATE_C smnC conf
```

первая вычисляет матрицу **S** на основании информации о моделируемой конфигурации, а вторая вычисляет значения матрицы воздействий, решает СЛАУ (нахождение матрицы σ) и вычисляет значения емкостной матрицы. Видно, что при этом доступ к матрице решения σ отсутствует. Поэтому в ходе работы необходимо вместо второй команды (CALCULATE_C) использовать следующие две команды

```
SET "SolVec" CALCULATE_SOLUTION_VECTOR smnC conf
SET "mC" CALCULATE_C_WITH_SOLUTION_VECTOR_ARGUMENT smnC conf SolVec
```

Первая из них вычисляет значения матрицы воздействий и решает СЛАУ (нахождение матрицы σ). Вторая вычисляет значения емкостной матрицы на основании только что (предыдущая команда) вычисленной матрицы решения.

Для останова итераций удобно воспользоваться операторов if. Далее пример его использования (синтаксис представляется интуитивно понятным, поэтому пояснения по его работе опущены)

```
IF LESS MINUS C C_old 0.01
THEN BREAK
```

2.7.6 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического

профиля. Общие требования и правила оформления".

2.8 Метод конечных элементов

Цель работы – получение навыков использования и программной реализации метода конечных элементов.

Для достижения поставленной цели используется программное обеспечение GNU Octave или Scilab. В ходе работы необходимо разработать программу для вычисления значений потенциалов.

2.8.1 Краткая теоретическая справка

Метод конечных элементов (МКЭ) считается наиболее мощным и универсальным численным методом решения задач, связанных со сложными геометриями и неоднородными средами, по сравнению с методами конечных разностей и моментов, которые концептуально проще и легче реализуемы в программном коде. Методическая общность метода позволяет строить на его основе универсальные компьютерные программы для решения широкого круга задач. Поэтому программы, разработанные для решения задач из других дисциплин, могут быть успешно применены для решения задач из другой предметной области с незначительными модификациями или без таковых.

Использование МКЭ при решении любой задачи в общем виде состоит из следующих шагов:

- дискретизация области решения на конечное число подобластей или конечных элементов, выбор базисных функций (В рассматриваемой области решения фиксируется конечное число точек, называемых узловыми точками (узлами). Значение непрерывной величины, которая должна быть определена в каждой узловой точке, считается переменным. Далее область определения непрерывной величины разбивается на конечное число подобластей, называемых элементами. Эти элементы имеют общие узловые точки и в совокупности аппроксимируют *форму области*);

- формирование разрешающих уравнений для элементарного элемента, формируется локальные матрицы (непрерывную величину аппроксимируют на каждом элементе полиномом (функцией элемента), который определяется с помощью узловых значений этой величины. Для каждого элемента подбирают свой полином таким образом, чтобы сохранить непрерывность величины вдоль границ элемента.);
- сбор всех элементов в области решения в конечно-элементную сетку (ансамблирование), формирование СЛАУ (Результаты аппроксимации подставляются в уравнения Максвелла (или эквивалентные им) с учётом граничных условий. В результате получается СЛАУ.);
- решение результирующего уравнения (СЛАУ);

определение интересующих (расчетных) величин в элементах.

2.8.2 Порядок выполнения работы

1. С помощью подхода, описанного в разделе 1.5.1 и программного кода, представленного в разделе 1.5.2 разработать программу для решения уравнения Лапласа двумерной области, показанной на рисунке 2.14 *а*, и разбитой на 25 треугольных конечных элемента (рисунок 2.14 *б*).
2. Сравнить полученные значения потенциалов с полученными методом конечных разностей: $\Phi_8 = 15{,}41$; $\Phi_9 = 26{,}74$; $\Phi_{10} = 56{,}69$; $\Phi_{13} = 34{,}88$; $\Phi_{14} = 65{,}41$; $\Phi_{17} = 58{,}72$.
3. Модернизировать программу для вычисления при разбиении конечных элементов 12 и 13 пополам.
4. Сравнить результаты, полученные в п. 2 и 3.
5. Сформировать выводы по работе.
6. Оформить отчет.

2.8.3 Содержание и требования к оформлению отчета

Отчет должен содержать титульный лист, название работы и цель работы, исходные данные, результаты расчетов, таблицы и графики, анализ результатов и выводы по работе.

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

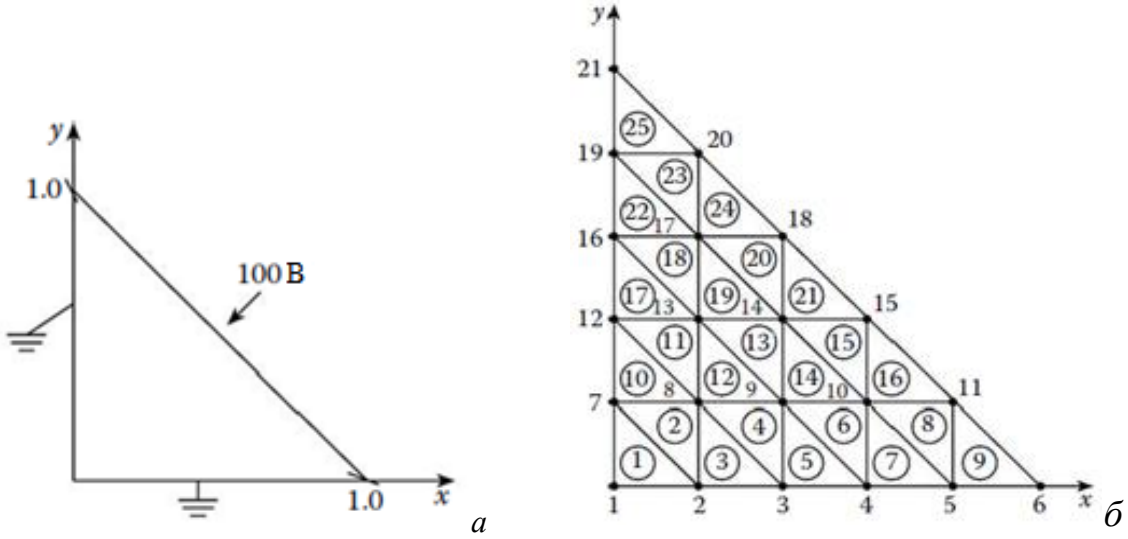


Рисунок 2.14 – Двумерная электростатическая задача (*а*)
и сетка, состоящая из 25 КЭ (*б*)

3 Самостоятельные работы

3.1 Общие положения

В процессе подготовки к практическим занятиям, студентам необходимо обратить особое внимание на проработку лекционного материала и самостоятельное изучение рекомендованной учебно-методической, а также научной и популярной литературы. Самостоятельная работа с учебниками, учебными пособиями, научной, справочной и популярной литературой, материалами периодических изданий и Интернета, статистическими данными является наиболее эффективным методом получения знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала, формирует у студентов свое отношение к конкретной проблеме. Более глубокому раскрытию вопросов способствует знакомство с дополнительной литературой, рекомендованной преподавателем по каждой теме семинарского или практического занятия, что позволяет студентам проявить свою индивидуальность в рамках выступления на данных занятиях, выявить широкий спектр мнений по изучаемой проблеме.

Отдельного внимания заслуживает изучение особенностей других численных методов, не рассматриваемых в ходе дисциплины.

Для получения больших навыков работы с программными средствами рекомендуется рассмотрение тестовых примеров и их усовершенствование, изучение новых функций и прочих особенной программной реализации, например, использование симметрии, разворачивание циклов и пр. Помимо этого требуется выполнение нескольких заданий, по результатам которых должны быть подготовлены соответствующие отчеты.

3.2 Методы решения СЛАУ

3.2.1 Влияние малых изменений матрицы и вектора свободных членов на процесс решения СЛАУ

Цель работы – оценка влияния малых изменений (возмущениях) матрицы и вектора свободных членов СЛАУ на процесс её решения.

Работа должна быть выполнена в GNU Octave или Scilab.

3.2.1.1 Последовательность выполнения работы

1. Сформировать произвольную квадратную матрицу **A**, $N = 6$.
2. Сформировать произвольный вектор **b**.
3. Найти решение системы **Ax** = **b** с помощью левого матричного деления (**x** = **A\b**).
4. Вычислить невязку **r** и ее норму.
5. Сформировать малое возмущение вектора **b** (**delta_b**).
6. Вычислить абсолютную и относительную нормы возмущения.
7. Вычислить возмущенный вектор свободных членов **bdb** = **b** + **delta_b**.
8. Найти возмущенное решение **xdb** и возмущение решения **delta_x** = **xdb** – **x**.
9. Вычислить абсолютную и относительную нормы возмущения решения.
10. Вычислить отношение относительных норм возмущения решения и вектора свободных членов.
11. Вычислить число обусловленности матрицы **A** и сравнить его с отношением норм, полученным в предыдущем пункте.
12. Полученные результаты оформить в виде отчета по самостоятельной работе.
13. Отчет должен также содержать аналогичный анализ не менее 4 СЛАУ при $10 \le N \le 100$, сопровождающийся приведением только скалярных значений (нормы и числа обусловленности).
14. Сформулировать выводы по работе.

3.2.1.2 Требования к оформлению отчета

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

3.2.2 Влияние машинной арифметики на точность решения

Цель работы – изучение особенностей машинной арифметики на примере решения СЛАУ.

Работа должна быть выполнена в GNU Octave или Scilab.

3.2.2.1 Последовательность выполнения работы

1. Сформировать произвольную квадратную матрицу **A**, $N = 6$.
2. Вычислить число обусловленности матрицы **A**.
3. Получить значение «машинного эпсилона» с помощью команды *eps* (%*eps* – Scilab).
4. Сформировать произвольное решение **x_exact**, которое далее считается точным.
5. Вычислить вектор свободных членов **b** = **Ax_exact**.
6. Найти приближенное решение системы **Ax** = **b** с помощью левого матричного деления (**x** = **A\b**).
7. Вычислить невязку r и её норму.
8. Вычислить ошибку приближенного решения **delta_x** = **x** – **x_exact** и относительную ошибку norm(**delta_x**)/norm(**x_exact**).
9. Выполнить сравнение полученного значения с произведением «машинного эпсилона» на число обусловленности матрицы **A**
10. Полученные результаты оформить в виде отчета по самостоятельной работе.
11. Отчет должен также содержать аналогичный анализ не менее 4 СЛАУ при $10 \le N \le 100$, сопровождающийся приведением только скалярных значений (нормы, числа обусловленности и их произведения на «машинный эпсилон»).
12. Сформировать выводы по работе.

3.2.2.2 Требования к оформлению отчета

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического

профиля. Общие требования и правила оформления".

3.3 Хранение разреженных матриц

Цель работы – изучение особенностей хранения разреженных матриц.

3.3.1 Определение разреженной матрицы

Известно, что для уменьшения вычислительных затрат в итерационных методах используют предобусловливание. Однако это требует хранения дополнительной матрицы, что увеличивает затраты на её хранение. Особенностью дополнительной матрицы является то, что она, как правило, является разреженной. Хранить её можно с помощью любого из форматов хранения разреженных матриц для экономии машинной памяти. Далее выполнен обзор этих форматов.

Разреженная матрица – матрица, имеющая преимущественно нулевые элементы. Объем ненулевых элементов, в общем случае, определяется для каждой отдельной задачи. Еще одно из определений: матрица порядка N, число ее ненулевых элементов должно выражаться как $N^{1+\gamma}$, где $\gamma < 1$. Матрица разрежена, если $\gamma \leq 0{,}2$ и для ленточных матриц, если $\gamma \leq 0{,}5$.

Другое определение заключается в том, что матрицу следует считать разреженной, если удалось извлечь выгоду из наличия в ней нулей. Такой выгодой может стать, например, уменьшение памяти компьютера для хранения матрицы в сжатом формате, в отличие от формата, не учитывающего наличие нулевых элементов.

3.3.2 Форматы хранения разреженных матриц

Существует много различных форматов хранения разреженных матриц. Они отличаются по степени сжатия матрицы и по затратам машинного времени, необходимого для элементарных операций с матрицей (например, поиск элемента, добавление нового элемента и др.). Как правило, форматы, обеспечивающие хорошее сжатие, требуют для операций больше машинного времени и наоборот. Поэтому требуется выбирать формат для каждой

конкретной задачи.

Из наиболее известных форматов следует выделить следующие: из двух векторов, Кнута, Рейнболдта и Местеньи, Ларкума, Шермана, разреженный строчный (Compressed Storage Row, CSR), разреженный столбцовый (Compressed Storage Column, CSC), разреженный неравномерный диагональный (Jagged Diagonal Storage, JDS), сжатый диагональный (Compressed Diagonal Storage, CDS), разреженный блочный строчный (Sparse Block Compressed Row Storage, SBCRS) и др.

Существуют форматы хранения разреженных матриц, предназначенные только для хранения определенного вида матриц (симметричных, ленточных и др.). Так, формат CDS предназначен для хранения только ленточных матриц. Далее рассмотрим самые распространенные форматы хранения разреженных матриц.

3.3.2.1 Формат из двух векторов

При использовании формата из двух векторов матрица хранится в виде двух векторов (**I** и **A**). Вектор **I** содержит номер столбца, **A** – значения ненулевых элементов. Если элемент в векторе **I** равен нулю, то это означает начало строки, при этом элемент вектора **A** содержит номер следующей строки. Если нули в двух векторах, то это означает конец матрицы. На рисунке 3.1 приведен пример формата.

	1	2	3	4	5
1	a_{11}			a_{14}	
2		a_{22}			a_{25}
3	a_{31}		a_{33}		a_{35}
4					
5	a_{51}				a_{55}

Вектор	Индекс элемента													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I	0	1	4	0	2	5	0	1	3	5	0	1	5	0
A	1	a_{11}	a_{14}	2	a_{22}	a_{25}	3	a_{31}	a_{33}	a_{35}	5	a_{51}	a_{55}	0

Рисунок 3.1 – Представление матрицы с помощью формата из двух векторов

Основным преимуществом формата является то, что он обеспечивает хороший коэффициент сжатия. Недостатком является сложность выполнения элементарных операций, таких как поиск и добавление элемента матрицы.

3.3.2.2 Формат Кнута

Формат предложен Кнутом в 1968 г. Матрица хранится в виде семи векторов, трех основных (**AN**, **I**, **J**) и четырех дополнительных (**NR**, **NC**, **JR**, **JC**). Первый вектор (**AN**) содержит ненулевые элементы матрицы в произвольном порядке. Второй (**I**) и третий (**J**) векторы содержат индексы столбцов и строк элемента, записанного в первом векторе. Для ускорения поиска ненулевого элемента использованы дополнительные четыре вектора, в которых хранятся указатели на позиции ненулевого элемента в векторе **AN**. **NR** – указатель на следующий элемент в строке (**NC** – в столбце) исходной матрицы, а **JR** – указатель на первый элемент в строке (**JC** – в столбце). На рисунке 3.2 приведен пример представления матрицы с помощью формата Кнута.

	1	2	3	4	5
1	a_{11}			a_{14}	
2		a_{22}			a_{25}
3	a_{31}		a_{33}		a_{35}
4					
5	a_{51}				a_{55}

Вектор	Индекс элемента								
	1	2	3	4	5	6	7	8	9
AN	a_{11}	a_{14}	a_{22}	a_{25}	a_{31}	a_{33}	a_{35}	a_{51}	a_{55}
I	1	4	2	5	1	3	5	1	5
J	1	1	2	2	3	3	3	5	5
NR	2	0*	4	0	6	7	0	9	0
NC	5	0	0	7	8	0	9	0	0
JR	1	3	5	0**	8				
JC	1	3	6	2	4				

Рисунок 3.2 – Представление матрицы с помощью формата Кнута
(* – текущий элемент последний в строке/столбце,
** – в данной строке нет ненулевых элементов)

Недостаток формата заключается в том, что для хранения каждого элемента матрицы требуется 5 значений (значения векторов **AN**, **I**, **J**, **NR**, **NC**), вследствие чего мал коэффициент сжатия. Достоинством формата является легкость добавления (или удаления) элементов матрицы, поэтому он хорошо подходит в алгоритмах, где нельзя предсказать конечное число ненулевых элементов (например, LU-разложение).

3.3.2.3 Формат Рейнболдта и Местеньи

Формат Рейнболдта и Местеньи (1973 г.) является модификаций формата

Кнута. В отличие, от которого векторы **NR** и **NC** закольцовываются, т.е. последний элемент в строке (в столбце) содержит указатель на первый элемент строки (столбца). Векторы **I, J** не хранятся. Формат требует значительно меньше памяти, однако при поиске элемента необходимо просматривать все элементы текущей строки (столбца), т.к. заранее неизвестны индексы её ненулевых элементов. Пример приведен на рисунке 3.3.

	1	2	3	4	5
1	a_{11}			a_{14}	
2		a_{22}			a_{25}
3	a_{31}		a_{33}		a_{35}
4					
5	a_{51}				a_{55}

Вектор	Индекс элемента								
	1	2	3	4	5	6	7	8	9
AN	a_{11}	a_{14}	a_{22}	a_{25}	a_{31}	a_{33}	a_{35}	a_{51}	a_{55}
NR	2	1	4	3	6	7	5	9	8
NC	5	2	3	7	8	9	8	1	4
JR	1	3	5	0**	8				
JC	1	3	6	2	4				

Рисунок 3.3 – Представление матрицы с помощью формата Рейнболдта и Местеньи (* – в данной строке нет ненулевых элементов)

3.3.2.4 Формат Ларкума

Усовершенствованный вариант формата Кнута использовался Ларкумом (1971 г.) для хранения симметричных (или треугольных) матриц (рисунок 3.4). Все элементы главной диагонали (даже если элемент нулевой) и ненулевые элементы матрицы хранятся в векторе **AN**. Еще в двух векторах хранятся указатели на ненулевые элементы: **NR** – описывает строку слева направо; **NC** – описывает столбец над диагональю снизу вверх. Если добавить еще два аналогичных вектора, то возможно хранить несимметричные матрицы.

	1	2	3	4	5
1	a_{11}			a_{14}	
2		a_{22}			a_{25}
3			a_{33}		a_{35}
4					
5					a_{55}

Вектор	Индекс элемента							
	1	2	3	4	5	6	7	8
AN	a_{11}	a_{22}	a_{33}	0*	a_{55}	a_{14}	a_{25}	a_{35}
NR	6	7	8	0**	0	0	0	0
NC	0**	0	0	6	8	0	0	7

Рисунок 3.4 – Представление матрицы с помощью формата Ларкума (* – диагональный элемент нулевой; ** – ноль в данном векторе означает, что текущий элемент последний в строке/столбце)

3.3.2.5 Формат Шермана

Еще один из форматов (рисунок 3.5) для треугольных матриц был предложен Шерманом (1975 г.). Формат состоит из пяти векторов. В **UD** – хранятся диагональные элементы, **UN** – недиагональные ненулевые, **IU** – указатели на первые элементы строк недиагональных элементов, **JU** – индексы столбцов, которые сжаты специальным образом (повторяющиеся индексы столбцов, для разных строк, записываются один раз, рисунок 3.6), **U** – указатели на первые элементы столбца вектора **JU**.

	1	2	3	4	5
1	a_{11}		a_{13}	a_{14}	a_{15}
2		a_{22}			a_{25}
3			a_{33}	a_{34}	a_{35}
4					
5					a_{55}

Вектор	Индекс элемента					
	1	2	3	4	5	6
UD	a_{11}	a_{22}	a_{33}	0	a_{55}	
UN	a_{13}	a_{14}	a_{15}	a_{25}	a_{34}	a_{35}
IU	1	4	5	0*	0*	
U	1	3	1	0	0	
JU**	3	4	5			

Рисунок 3.5 – Представление матрицы с помощью формата Шермана (* – в данной строке нет элементов; ** – правило сжатия представлено на рисунке 3.6)

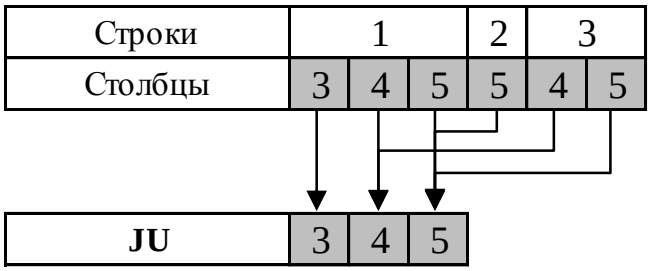


Рисунок 3.6 – Сжатие вектора JU

3.3.2.6 Формат CSR (CSC)

Форматы CSR и CSC, предложенные Чангом (1969 г.) и Густавсоном (1972 г.), широко используются для хранения разреженных матриц. Эти форматы предъявляют минимальные требования к памяти и эффективны для реализации нескольких важных операций над разреженными матрицами: сложения, умножения, перестановок строк и столбцов, транспонирования, решения СЛАУ с разреженными матрицами как прямыми, так и итерационными методами и т.д. В формате CSR значения ненулевых элементов

матрицы и соответствующие им индексы столбцов хранятся по строкам в двух векторах (**Values** – ненулевые элементы и **Columns** – индексы столбцов). Используется также вектор указателей на ненулевые элементы (**RowIndex**), с которых начинается очередная строка (рисунок 3.7). Формат CSC отличается только порядком хранения ненулевых элементов матрицы: в CSR элементы хранятся по строкам, в CSC – по столбцам. При этом используются векторы **Values**, **Rows**, **ColumnsIndex**, – соответственно (рисунок 3.8).

	1	2	3	4	5
1	a_{11}			a_{14}	
2		a_{22}			a_{25}
3	a_{31}		a_{33}		a_{35}
4					
5	a_{51}				a_{55}

Вектор	Индекс элемента								
	1	2	3	4	5	6	7	8	9
Values	a_{11}	a_{14}	a_{22}	a_{25}	a_{31}	a_{33}	a_{35}	a_{51}	a_{55}
Columns	1	4	2	5	1	3	5	1	5
RowIndex	1	3	5	0*	8				

Рисунок 3.7 – Представление матрицы с помощью формата CSR
(* – в данной строке нет элементов)

	1	2	3	4	5
1	a_{11}			a_{14}	
2		a_{22}			a_{25}
3	a_{31}		a_{33}		a_{35}
4					
5	a_{51}				a_{55}

Вектор	Индекс элемента								
	1	2	3	4	5	6	7	8	9
Values	a_{11}	a_{31}	a_{51}	a_{22}	a_{33}	a_{14}	a_{25}	a_{35}	a_{55}
Rows	1	3	5	2	3	1	2	3	5
ColumnsIndex	1	4	5	6	7				

Рисунок 3.8 – Представление матрицы с помощью формата CSC

3.3.2.7 Формат хранения JDS

На первом шаге сжатия все ненулевые элементы матрицы сдвигаются влево, игнорируя нулевые, и хранятся в матрице **Val**. Номера столбцов этих элементов хранятся в отдельной матрице **Col**. Если в строке содержатся только нулевые элементы, то в соответствующих строках матрицы **Val** и **Col** хранятся нули (рисунок 3.9).

Однако при таком способе хранения возникает избыточность, которая устраняется дополнительным сжатием матрицы **Val** по столбцам. В результате получаются три вектора (см. рисунок 3.10): **Values** (значения ненулевых элементов), **Columns** (индекс столбца ненулевого элемента) **StartPosition**

(указатель на начальный элемент в столбце матрицы **Val**).

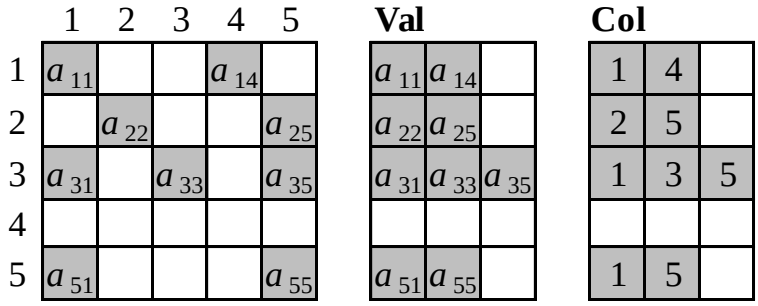
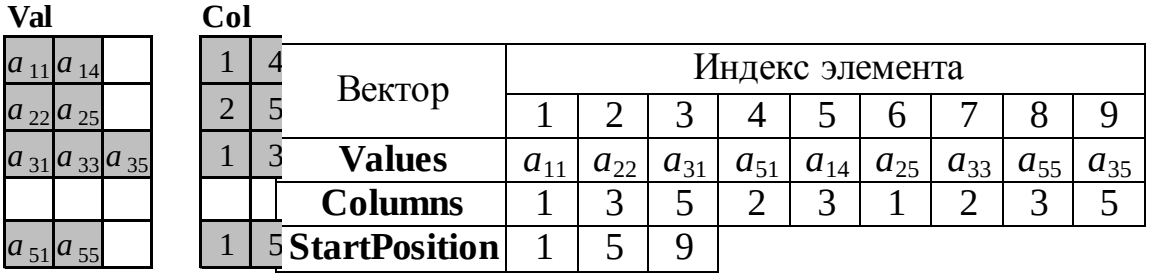


Рисунок 3.9 – Первый шаг сжатия матрицы с помощью формата JDS



Вектор	Индекс элемента								
	1	2	3	4	5	6	7	8	9
Values	a_{11}	a_{22}	a_{31}	a_{51}	a_{14}	a_{25}	a_{33}	a_{55}	a_{35}
Columns	1	3	5	2	3	1	2	3	5
StartPosition	1	5	9						

Рисунок 3.10 – Представление матрицы с помощью формата JDS

3.3.2.8 Сжатый диагональный формат хранения (CDS)

Сжатый диагональный формат (CDS) предназначен для хранения ленточных матриц (матрицы, у которых ненулевые элементы расположены только на главной диагонали и примыкающих к ней. Исходная матрица представляется матрицей **Val**, где диагонали исходной матрицы хранятся по строкам, начиная слева направо (см. рисунок 3.11). При этом нет необходимости хранить дополнительную информацию.

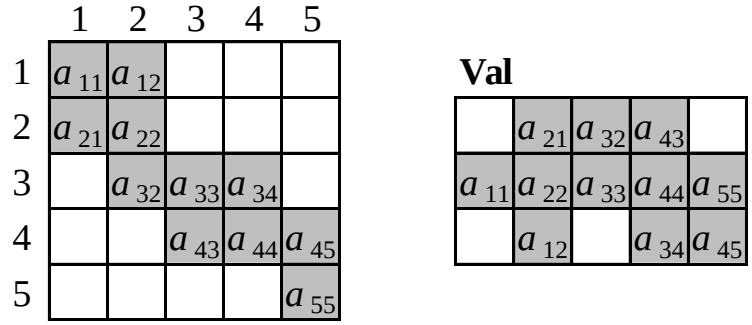


Рисунок 3.11 – Представление матрицы с помощью формата CDS

3.4 Реализация форматов хранения

Цель работы – изучение особенностей программной реализации форматов хранения разреженных матриц CRC и CDS.

Работа должна быть выполнена в GNU Octave или Scilab.

3.4.1 Последовательность выполнения работы

1. Разработать программу для реализации формата хранения CRC.
2. Продемонстрировать правильность программной реализации на примере из подраздела 3.3.2.
3. Сгенерировать разреженную матрицу **A** порядка $N = 100$.
4. Выполнить оценку затрат машинной памяти для хранения матрицы A с помощью стандартного и реализованного формата хранения.
5. Полученные результаты оформить в виде отчета по самостоятельной работе.
6. Сформулировать выводы по работе.

3.4.2 Требования к оформлению отчета

Оформление должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

3.5 Инструментальные средства

На рынке программных средств для моделирования имеется целый ряд систем, имеющих специализированные студенческие или демо-версии, которые предназначены для ознакомления с особенностями их использования. Данные версии, как правило, имеют неограниченный срок использования и ряд ограничений, касающихся вычислительных возможностей (численный метод, методы решения СЛАУ, способ дискретизации и пр.).

В таблице 3.1 приведен перечень инструментальных средств, подлежащих изучению в ходе самостоятельной работы. Используемые версии данных средств доступны для скачивания на официальном сайте производителя.

Цель работы – изучение особенностей инструментальных средств и работы в них.

Таблица 3.1 – Перечень инструментальных средств, подлежащих изучению

№ п.п.	Инструментальное средство	Сайт разработчика
1.	CST STUDIO SUITE Student version	https://www.cst.com
2.	FEKO LITE	https://altairhyperworks.com
3.	Concept-II demo version	http://www.tet.tuhh.de
4.	newFasant Silver Version	https://www.fasant.com
5.	QuickWave STUDENT Release	http://www.qwed.com.pl
6.	Sonnet Lite	http://www.sonnetsoftware.com
7.	WipL-D Demo	https://www.wipl-d.com

По каждому инструментальному средству необходимо подготовить отчет о проделанной работе, в котором должны быть отражены предметные области применения средства, особенности работы в нем, перечень доступных для анализа численных методов и методов решения СЛАУ, аппаратных ускорителей, а также способов задания дискретизации. Отчет может содержать и другую информацию, имеющую непосредственное отношение, по мнению исполнителя, к изучаемой дисциплине.

Оформление отчета должно соответствовать ОС ТУСУР 01-2013 "работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления".

Список использованных источников

1. Григорьев А.Д. Методы вычислительной электродинамики – М.: Физматлит, 2013. – 430 с.
2. Алексеев Е.Р. GNU Octave для студентов и преподавателей / Е.Р. Алексеев, О.В. Чеснокова // Донецк.: ДонНТУ, Технопарк ДонНТУ УНИТЕХ. –2011. – С. 332.
3. Алексеев Е.Р. Scilab: Решение инженерных и математических задач / Е.Р. Алексеев, О.В. Чеснокова, Е.А. Рудченко. – М.: ALT Linux; БИНОМ. Лаборатория знаний, 2008. – 260 с.
4. Sadiku M.N.O. Numerical. Techniques in Electromagnetics. Third Edition. – CRC Press LLC, 2009. – 710 p.
5. Computational Electromagnetics / T. Rylander, P. Ingelström, A. Bondeson. – New York: Springer Science+Business Media, 2013. – 286 p.
6. Гринев А.Ю. Математические основы и методы решения задач электродинамики / А.Ю. Гринев, А.И. Гиголо. – Москва: Радиотехника, 2015. – 216 с.
7. Dworsky L.N. Introduction to numerical electrostatics using MATLAB. – Wiley, 2014. – 436 p.
8. Gibson W.C. The Method of Moments in Electromagnetics, Second Edition. – Chapman and Hall/CRC, 2014. – 450 p.
9. Schlagenhaufer F., He J., Fynn K. Computer Simulation and Experimental Validation of a Metallic Enclosure with Slots // IEEE Int. Symp. on EMC, 2002, pp. 273–278.
10. Hubing T.H. An EMC engineer's guide to electromagnetic modeling software // IEEE EMC Society Distinguished Lecturer Presentation. – Ottawa, 1997. – P. 1–21.
11. Su C. Overview of electromagnetic modeling software / C. Su, H. Ke, T. Hubing // Proc. of the 25th International Review of Progress in Applied Computational Electromagnetics. – Monterey, CA, March 2009. – P. 1–6.

12. Hubing T.H. How EMC engineers use computer modeling tools productively // First International Conference on Automation, Control, Energy and Systems (ACES), plenary lecture. – 2014. – P. 1–56.
13. Банков С.Е. Электродинамика и техника СВЧ для пользователей САПР / С.Е. Банков, А.А. Курушин // М. Солон-Пресс. – 2010. – С. 276.
14. Банков С.Е. Электродинамическое моделирование антенных и СВЧ-структур с использованием FEKO / С.Е. Банков, А.Н. Грибанов, А.А. Курушин // М. OneBook. – 2013. – С. 426.
15. Standard IEEE 1597.2. Recommended Practice for Validation of Computational Electromagnetics Computer Modeling and Simulations. – 2010.
16. Standard IEEE 1597.2. Recommended Practice for Validation of Computational Electromagnetics Computer Modeling and Simulations. – 2010.
17. Jakobus U. Aspects of and insights into the rigorous validation, verification, and testing processes for a commercial electromagnetic field solver package / U. Jakobus, R.G. Marchand, D.J. Ludick // IEEE Transactions on Electromagnetic Compatibility. – 2014. – Vol. 56, Iss. 4. – P. 759–770.
18. Lessons from applying IEEE standard 1597 for validation of computational electromagnetics computer modeling and simulations / S. Park, M. Kotzev, H.D.D. Bruns, D.G. Kam, C. Schuster // IEEE Electromagnetic Compatibility Magazine. – 2017. – Vol. 6, No. 2. – P. 55–67.
19. Development of next generation FSV tools and standards / A.L. Drozd, B. Archambeault, A. Duffy, I. Kasperovich // IEEE International Symposium on Electromagnetic Compatibility. – Pittsburgh, PA, USA, August 6–10, 2012. – P. 647–648.
20. Geng Y. Research on FSV in membership function credibility verification – for system performance evaluation / Y. Geng, T. Jiang, X. Zhang // Proc. of progress in electromagnetic research symposium (PIERS). – Shanghai, China, August 8–11, 2016. – P. 4723-4727.
21. Многократное решение систем линейных алгебраических уравнений итерационными методами с предобусловливанием в задачах

электромагнитной совместимости / Р.Р. Ахунов, С.П. Куксенко, Т.Р. Газизов, П.Е. Орлов. – Томск: Изд-во Томск. гос. ун-та систем упр. и радиоэлектроники, 2015. – 152 с.

22. Куксенко С.П. Итерационные методы решения системы линейных алгебраических уравнений с плотной матрицей / С. П. Куксенко, Т.Р. Газизов. – Томск: Томский государственный университет, 2007. – 208 с.
23. Совершенствование моделирования и обеспечения электромагнитной совместимости бортовой радиоэлектронной аппаратуры космических аппаратов: моногр. / В.К. Салов, А.М. Заболоцкий, С.П. Куксенко, П.Е. Орлов, Р.С. Суровцев. Томск: изд-во Томск. гос. ун-та систем упр. и радиоэлектроники, 2014. – 131 С.
24. Rjasanow S. The Fast Solution of Boundary Integral Equations (Mathematical and Analytical Techniques with Applications to Engineering) / S. Rjasanow, O. Steinbach. – New York: Springer-Verlag, 2007.