Министерство образования и науки Российской Федерации
Томский государственный университет
систем управления и радиоэлектроники

Многократное решение систем линейных алгебраических уравнений итерационными методами с предобусловливанием в задачах электромагнитной совместимости

Монография

Томск
Издательство ТУСУРа
2015

УДК 519.612
ББК 22.193
М73

Рецензент
Дмитренко А.Г., д-р физ.-мат. наук, профессор
Национального исследовательского
Томского государственного университета

Авторы:
Р.Р. Ахунов, С.П. Куксенко, Т.Р. Газизов, П.Е. Орлов

Исследования выполнены за счет проектов
РФФИ 14-07-31267 и 14-29-09254 (алгоритмы),
госзадания 8.1702.2014/К (программы),
РНФ 14-15-01232 (моделирование)

М73 **Многократное** решение систем линейных алгебраических уравнений итерационными методами с предобусловливанием в задачах электромагнитной совместимости : моногр. / Р.Р. Ахунов [и др.]. – Томск: Изд-во Томск. гос. ун-та систем упр. и радиоэлектроники, 2015. – 152 с.

ISBN 978-5-86889-720-7

Приведен обзор задач имитационного моделирования в электромагнитной совместимости. Описаны прямые и итерационные методы решения системы линейных алгебраических уравнений, а также форматы хранения разреженных матриц. Предложены алгоритмы итерационных методов с предобусловливанием, использующие разреженные форматы хранения матриц, и алгоритмы многократного решения систем линейных алгебраических уравнений. Представлены результаты экспериментов на примере решения задач вычисления емкостной матрицы различных полосковых структур. Приведены исходные коды программ на языке C++.

Для студентов и аспирантов физико-математических, экономических и инженерных специальностей.

УДК 519.612
ББК 22.193

ISBN 978-5-86889-720-7

Введение

Невыполнение требований электромагнитной совместимости (ЭМС) может повлечь серьезные последствия в различных сферах деятельности человека. Облегчить обеспечение ЭМС позволяет предварительное имитационное моделирование. В его основе лежит численный анализ посредством решения уравнений Максвелла, для которого широко используется метод моментов (МоМ). Один из его самых трудоемких этапов приходится на решение системы линейных алгебраических уравнений (СЛАУ) с плотной матрицей, поэтому актуально совершенствование методов решения СЛАУ.

Методы решения СЛАУ делят на две большие группы: прямые (или точные) и итерационные методы. К прямым относятся методы, которые позволяют получить решение за конечное число элементарных операций. Итерационные методы, в отличие от прямых, основаны на последовательном приближении к решению СЛАУ. Основными преимуществами итерационных методов являются более низкие вычислительные затраты и меньшая погрешность при вычислении плохо обусловленных СЛАУ.

В итерационных методах для уменьшения числа итераций используют предобусловливание. Для этого необходимо хранить дополнительную матрицу, что увеличивает требуемую память компьютера. Эта матрица, как правило, является разреженной. Поэтому хранить ее можно с помощью любого из форматов хранения разреженных матриц для экономии машинной памяти и в перспективе времени решения СЛАУ.

Проблема многократного решения СЛАУ возникает во многих научных и инженерных задачах. Примером такой задачи является вычисление емкостных матриц полосковых структур в диапазоне изменения их параметров. Эти задачи могут решаться итерационными методами с предобусловливанием, причем предобусловливатель, полученный для одной матрицы, можно использовать для других.

В предлагаемой книге рассматриваются алгоритмы итерационных методов решения СЛАУ в задачах электромагнитной совместимости.

В разделе 1 содержится краткое обоснование актуальности предварительного имитационного моделирования. Приведен порядок расчета емкостной матрицы методом моментов для произвольной двумерной структуры. Также сделан краткий обзор методов решения СЛАУ. Рассмотрены форматы хранения разреженных матриц и существующие варианты многократного решения СЛАУ.

В разделе 2 проведен сравнительный анализ наиболее распространенных форматов хранения разреженных матриц. Разработаны алгоритмы ILU(0)-разложения с применением формата хранения разреженных матриц CSR. Описан вычислительный эксперимент, подтверждающий эффективность использования разреженного формата.

В разделе 3 представлены разработанные алгоритмы для многократного решения СЛАУ. Приведен алгоритм, использующий итерационный метод с предобусловливанием для многократного решения СЛАУ. Рассмотрены различные способы, позволяющие адаптивно переформировывать предобусловливатель. Кроме того, приведены алгоритмы, ускоряющие решение без переформирования, а лишь за счет выбора очередности решаемых СЛАУ. По всем предложенным алгоритмам осуществлены вычислительные эксперименты, показавшие на различных полосковых структурах эффективность алгоритмов.

В разделе 4 приведены расчеты сложности алгоритмов LU-разложения и итерационных методов BiCGStab и CGS. Представлены исходные коды программ, разработанных на основе алгоритмов, предложенных в разделах 2 и 3.

Авторы глубоко благодарны Лилии Юрьевне Колотилиной за ее ранние математические работы, используемые авторами в своей научной деятельности; за обсуждение нескольких статей авторов, значительно улучшившее их содержание; за критику, подвигнувшую авторов к получению некоторых теоретических результатов; за моральную поддержку, вдохновившую авторов на развитие первых результатов исследования до данной монографии и позволяющую с оптимизмом смотреть в будущее.

1. Обзор актуальных задач моделирования

1.1. Актуальность моделирования для обеспечения электромагнитной совместимости

Электромагнитная совместимость (electromagnetic compatibility — EMC) — это способность технического средства (ТС) эффективно функционировать с заданным качеством в определенной электромагнитной обстановке (ЭМО), не создавая при этом недопустимых электромагнитных помех другим ТС [1]. Невыполнение требований ЭМС может иметь серьезные последствия в различных сферах деятельности человека и на производственных предприятиях: приводить к сбою в электронных системах управления воздушным, железнодорожным транспортом, автоматических производственных линиях, системах управления промышленными объектами и объектами энергетики, в работе медицинского оборудования и т.д. ЭМС нарушается, если уровень помех слишком высок или помехоустойчивость оборудования недостаточна.

Актуальность обеспечения электромагнитной совместимости обусловлена ее междисциплинарным характером, поскольку электрические и радиоэлектронные устройства используются в самых различных областях техники. Другой причиной является ее воздействие на всех структурных уровнях аппаратуры: от чипов до высоковольтных линий электропередачи. Наконец, актуальность обеспечения ЭМС диктуется тем, что она связана со сложными, часто скрытыми от традиционных знаний электромагнитными процессами, нарушающими обычное поведение систем. Борьба с помехами в радиотехнических системах различного назначения особо актуальна, причем не только с непреднамеренными (для обеспечения безопасности) [2], но и с преднамеренными (для защиты от угроз электромагнитного терроризма) [3], что в последние годы стало отдельной темой каждого международного симпозиума по ЭМС. Важность исследований в области ЭМС радиоэлектронной аппаратуры (РЭА) подтверждается активной деятельностью в этом направлении, причем в разных секторах (академическом, университетском, отраслевом) инженерных наук,

а также известных школ, которыми руководят В.Е. Фортов (Россия, РАН), Л.Н. Кечиев (Россия, ВШЭ-МГИЭМ), С.Ф. Чермошенцев (Россия, КГТУ им. А.Н. Туполева), С.А. Сухоруков (Россия, ЗАО «ЭМСОТЕХ»), E. Schamiloglu (Университет Нью Мексико, США), J.L. ter Haseborg (Германия, Гамбургский технологический университет), S. Tkachenko (Германия, Магдебургский университет), F. Rachidi (Швейцария, Политехнический университет Лозанны), W. Radasky (США, корпорация Metatech, МЭК).

С ростом быстродействия полупроводниковых приборов все большая доля времени задержки распространения сигналов приходится на задержки в межсоединениях электронных схем, ставшие существенным фактором, влияющим на быстродействие схемы в целом. Так, по данным Sematech [4], в скоростных полупроводниковых чипах задержки в межсоединениях составляют 80 % цикла, тогда как задержки переключения транзисторных ключей занимают лишь 20 % общего времени. В платах и блоках этот эффект проявляется еще сильнее, поскольку их размеры больше и длина межсоединений может составлять несколько длин волн распространяющихся по ним сигналов. Поэтому необходимо решать проблемы межсоединений, так как именно они зачастую становятся главной преградой на пути создания быстродействующей, компактной и в то же время помехоустойчивой и надежной аппаратуры. Неучет факторов, составляющих проблему, при проектировании какой-либо части устройства способен стать причиной сбоев и ненадежности в работе устройства в целом, которые трудно локализовать и устранить без больших затрат [5].

При проектировании РЭА выполняют предварительное имитационное моделирование, которое позволяет снизить финансовые и временные затраты на проведение повторных испытаний. Особая актуальность аспектов численного электромагнитного моделирования и оптимизации в связи с ростом частот подтверждается организацией в 2014 г. новой международной конференции «Численное электромагнитное моделирование и оптимизация для радиочастотных, сверхвысокочастотных и терагерцовых приложений» (NEMO2014), которая проводится ежегодно в Европе, Северной Америке и Азии. Задача обеспечения ЭМС РЭА вынуждает разработчиков проводить длительные дорогостоящие ис-

пытания. Устранение выявленных недостатков ведет к нарушениям рабочего графика и дополнительным финансовым затратам. Ранний и регулярный учет ЭМС в конструкции изделия минимизирует себестоимость и задержки проектирования, которые стали бы необходимыми в случае игнорирования вопросов ЭМС. Поэтому целесообразен учет ЭМС на этапе проектирования РЭА посредством имитационного моделирования с помощью специализированного программного обеспечения (ПО), основанного на схемотехническом, квазистатическом и электродинамическом подходах.

В общем случае в основе имитационного моделирования лежит численный анализ, требующий построения математической модели исследуемого объекта с помощью решения уравнений Максвелла. Численные методы, применяемые в моделировании ЭМС: методы конечных разностей во временной области [6]; метод моментов (МоМ) [7]; метод конечных элементов [8]; метод конечного интегрирования [9]; метод матрицы линий передачи [10], а также так называемые гибридные методики [11].

Процесс построения математической модели можно разбить на следующие этапы [12].

1. Постановка задачи — определение целей расчета, объема необходимой входной и выходной информации и допустимой погрешности.

2. Аналитическая обработка — формулировка уравнений Максвелла, начальных и граничных условий, описание параметров расчетной области, выбор метода решения, преобразование уравнений к виду, наиболее подходящему для данного численного метода.

3. Дискретизация (сегментация) модели — переход от непрерывных функций к дискретным и от функциональных уравнений к системе линейных алгебраических уравнений.

4. Вычисление элементов СЛАУ численным интегрированием или дифференцированием в зависимости от используемого на предыдущем этапе метода.

5. Решение СЛАУ — выбор наиболее подходящего метода решения (прямой или итерационный).

6. Обработка результатов (вычисление требуемых характеристик) — вычисление поля, погонных матриц, отклика и прочих характеристик и параметров исследуемого объекта/системы по данным из решения СЛАУ.

Важно, что перечисленные этапы не являются независимыми. Так, выбор метода дискретизации определяет затраты на формирование СЛАУ и влияет на размер и свойства получаемой СЛАУ, что в свою очередь определяет выбор метода ее решения. От предыдущих этапов зависят способы вычисления параметров и характеристик исследуемого объекта/системы. На втором этапе широко используются электродинамический и квазистатический (TEM-аппроксимация) подходы к решению уравнений Максвелла. На третьем этапе при решении задач ЭМС применяется метод моментов, использующий «поверхностный» подход, в соответствии с которым в качестве неизвестного выступает распределение плотности поверхностного тока на проводящих поверхностях исследуемого объекта/системы [13]. Найденный поверхностный ток рассматривается как источник, возбуждающий поле во всей расчетной области. Таким образом, при использовании МоМ неизвестная функция определена на поверхности, а не в объеме (как, например, в методах конечных разностей и элементов), что уменьшает требования к вычислительным ресурсам. Получаемая при этом матрица СЛАУ является плотной и плохо обусловленной, что требует построения эффективных предобусловливателей для ускорения решения посредством итерационных методов. Один из самых трудоемких этапов приходится на решение СЛАУ [5]. Поэтому актуально совершенствование методов решения СЛАУ.

В развитие методов решения СЛАУ сделали вклад такие отечественные и зарубежные ученые, как В.В. Воеводин, С.К. Годунов, В.П. Ильин, Л.Ю. Колотилина, Г.И. Марчук, В.В. Радченко, А.А. Самарский, Е.Е. Тыртышников, M. Bebendorf, S. Borm, C. Calgaro, J. Dongarra, D. Golub, W. Hackbusch, S. Karimi, Qing He, S. Rjasanow, Y. Saad, T. Topa, I. Tsukerman, Van der Vorst и др.

1.2. Вычисление емкостных матриц методом моментов

При использовании метода моментов рассматривается операторное уравнение, для решения которого применяют систему базисных функций в области определения оператора. Затем задается система весовых, или тестовых, функций в области значений оператора и берется скалярное произведение с каждой функцией. В результате получается матрица размером $N \times N$. На следующем шаге вычисляются элементы вектора воздействий размером N, таким образом формируется СЛАУ. Далее решается СЛАУ. Из вектора решения СЛАУ вычисляются требуемые характеристики моделируемой структуры. Одной из таких задач является вычисление методом моментов емкостных матриц произвольных двумерных [14] и трехмерных [15] структур проводников и диэлектриков.

Методом моментов решаются уравнения Максвелла в интегральной форме в частотной области. Реализация данного подхода на практике становится невозможной из-за крайне высоких требований к ресурсам компьютера, поэтому прибегают к различным упрощениям. Уравнения Максвелла сводят к частному случаю эллиптического дифференциального уравнения с частными производными, известного как уравнение Пуассона – Лапласа (полагается, что все заряды и токи сосредоточены на поверхности проводников, как в случае бесконечной проводимости проводников) с соответствующими граничными условиями [16].

Задача вычисления емкостной матрицы сводится к решению СЛАУ вида

$$\mathbf{S}\boldsymbol{\sigma} = \mathbf{V}, \tag{1.1}$$

где $\mathbf{S}$ — матрица, связывающая плотность заряда элементов дискретизации на проводниках и диэлектрических границах, составляющих вектор $\boldsymbol{\sigma}$, с потенциалами этих элементов, составляющими вектор $\mathbf{V}$. Система решается N_{cond} раз (N_{cond} — число проводников в системе, не считая опорного).

Для вычисления элементов матрицы $\mathbf{S}$ границы проводник-диэлектрик и диэлектрик-диэлектрик делятся на подынтервалы,

характеризуемые величинами: x_n — координата X центра подынтервала n; y_n — координата Y центра подынтервала n; dn — длина подынтервала n; θ_n — угол, образуемый подынтервалом n с положительным направлением оси координат X; ε_n — диэлектрическая проницаемость около n-го подынтервала проводник-диэлектрик; ε_n^+ и ε_n^- — диэлектрические проницаемости соответственно на положительной (к которой указывает $\underline{n}_n$) и отрицательной (от которой указывает $\underline{n}_n$) сторонах n-го подынтервала диэлектрик-диэлектрик, где $\underline{n}_n$ — единичный вектор, проведенный нормально от центра n-го подынтервала. Примеры значений этих параметров показаны в рамках на рисунке 1.1.

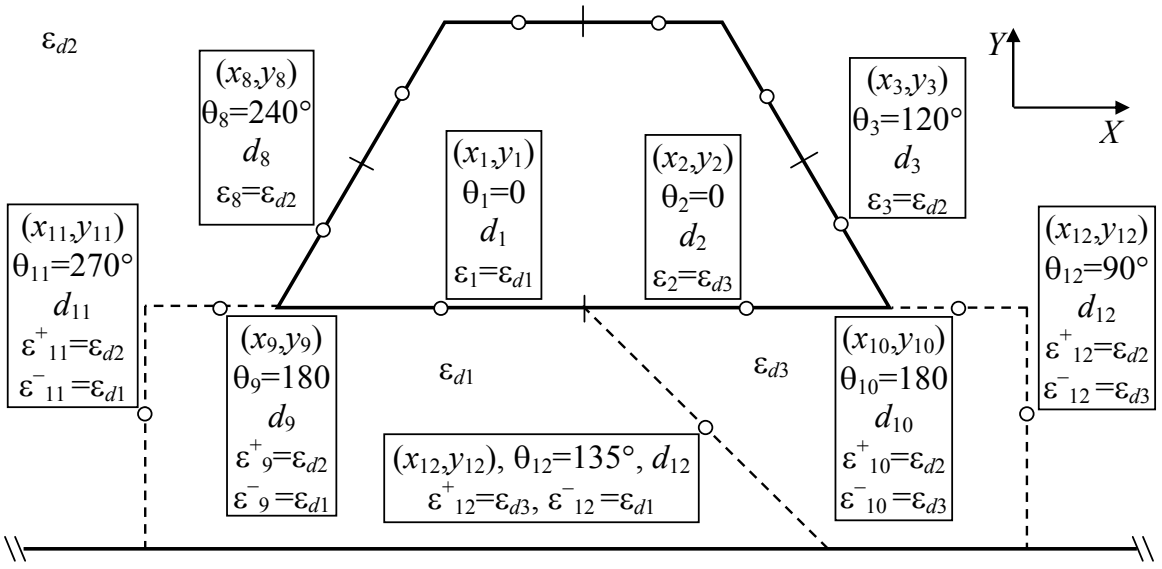


Рисунок 1.1 – Схема двумерной дискретизации

Центру подынтервала n соответствует вектор $\underline{r}_n$, определяемый как

$$\underline{r}_n = \underline{x}x_n + \underline{y}y_n, \tag{1.2}$$

где $\underline{x}$ и $\underline{y}$ — единичные векторы в направлениях X и Y соответственно.

Аналогично вектор $\underline{r'}_n$ подынтервала, по которому ведется интегрирование, и вектор $\underline{\hat{r}'}_n$ его образа относительно бесконечной плоскости определяются как

$$\underline{r}'_n = \underline{x}x'_n + \underline{y}y'_n; \tag{1.3}$$

$$\underline{\hat{r}}'_n = \underline{x}x'_n - \underline{y}y'_n; \tag{1.4}$$

$$x'_n = x_n + t\cos(\theta_n); \tag{1.5}$$

$$y'_n = y_n + t\sin(\theta_n), \tag{1.6}$$

где t — текущее расстояние от центра (x_n, y_n) подынтервала вдоль этого подынтервала.

В результате исходное интегрирование по двум декартовым координатам для текущей точки и ее образа сводится к интегрированию по одной переменной t (поскольку угол $\theta = \theta_n = \text{const}$) для выбранного подынтервала.

Порядок дискретизации таков. Сначала дискретизируются границы проводник-диэлектрик и полученным подынтервалам проводник-диэлектрик присваиваются номера с 1 по N_c. Если есть другие проводники, которые всегда находятся под нулевым потенциалом, то все они дискретизируются как обычные проводники. Затем дискретизируются границы диэлектрик-диэлектрик и полученным подынтервалам диэлектрик-диэлектрик присваиваются номера с $N_c + 1$ по N.

Для строк матрицы **S** с номерами $m = 1 \ldots N_c$, соответствующими подынтервалам проводник-диэлектрик, элементы S_{mn} вычисляются по формуле

$$S_{mn} = \frac{1}{2\pi\varepsilon_0}\left(\text{iflg} \cdot \hat{I}_{mn} - I_{mn}\right), \quad \begin{cases} m = 1,\ldots,N_c \\ n = 1,\ldots,N \end{cases}, \tag{1.7}$$

где

$$I_{mn} = \int_{\ell_n} \ln\left|\underline{r}_m - \underline{r}'_n\right| d\ell' ; \tag{1.8}$$

$$\hat{I}_{mn} = \int_{\ell_n} \ln\left|\underline{r}_m - \underline{\hat{r}}'_n\right| d\ell' . \tag{1.9}$$

Для строк матрицы **S** с номерами $m = (N_c + 1) \ldots N$, соответствующими подынтервалам диэлектрик-диэлектрик, элементы вычисляются по формулам

$$S_{mn} = \frac{1}{2\pi\varepsilon_0}\left(I_{mn} - \mathrm{iflg}\cdot\hat{I}_{mn}\right), \begin{cases} m=(N_c+1)...N \\ n=1...N \end{cases}, m \neq n; \quad (1.10)$$

$$S_{mm} = \frac{1}{2\pi\varepsilon_0}\left(I_{mm} - \mathrm{iflg}\cdot\hat{I}_{mm}\right) + \frac{1}{2\varepsilon_0}\frac{\varepsilon_m^+ + \varepsilon_m^-}{\varepsilon_m^+ - \varepsilon_m^-}, m=(N_c+1)...N, \quad (1.11)$$

где

$$I_{mn} = \int_{\ell_n} \frac{\underline{r}_m - \underline{r}'_n}{\left|\underline{r}_m - \underline{r}'_n\right|^2}\underline{n}_m d\ell'; \quad (1.12)$$

$$\hat{I}_{mn} = \int_{\ell_n} \frac{\underline{r}_m - \hat{\underline{r}}'_n}{\left|\underline{r}_m - \hat{\underline{r}}'_n\right|^2}\underline{n}_m d\ell'. \quad (1.13)$$

Следующим шагом является подстановка формул (1.2)–(1.6) в (1.7)–(1.13).

Исходя из приведенных формул, можно сделать вывод о структуре матрицы **S**. Условно матрицу **S** можно разделить на подматрицы, причем каждая подматрица будет соответствовать определенным подынтервалам. Для каждой подматрицы определены свои формулы вычисления элементов согласно выражениям (1.7), (1.10) и (1.11) (рис. 2).

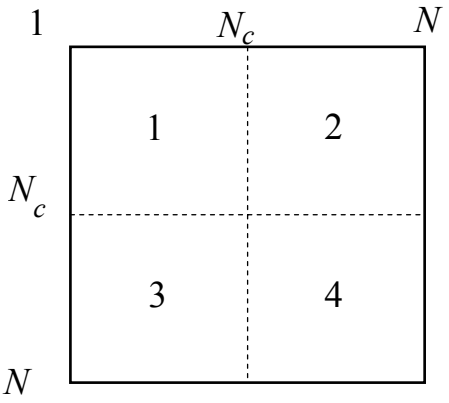


Рисунок 1.2 – Структура матрицы **S**

Подробный расчет для каждой области матрицы **S** можно найти в [16]. Из приведенных данных следует, что матрица **S** является плотной. Также можно сделать выводы о влиянии изменений параметров структуры на матрицу **S**. Так, при изменении ди-

электрической проницаемости ε (согласно формуле (1.11)) будут изменяться элементы матрицы **S** на главной диагонали нижней правой подматрицы с индексами элементов матрицы больше N_c (область 4 на рисунке 1.2). При изменении линейных размеров проводника могут изменяться все элементы матрицы **S**.

После вычисления вектора **σ** N_{cond} раз можно рассчитать матрицу емкостей. Каждый элемент емкостной матрицы определяется следующим образом:

$$C_{ij} = \int_{S_i} \frac{\varepsilon(\underline{r})}{\varepsilon_0} \sigma^{(j)}(\underline{r}) da_i,$$

где индекс *i* относится к проводнику, по поверхности которого производится интегрирование, а индекс *j* над плотностью заряда означает распределение плотности заряда, когда *j*-й проводник находился под потенциалом 1 В, а все остальные — под потенциалом нуля.

На практике часто требуется для получения точных результатов учет частотной зависимости ε [17]. При этом решение СЛАУ выполняется для каждой частотной точки из диапазона, что увеличивает общее время вычисления пропорционально числу частотных точек. Однако, как сказано выше, изменяются не все, а только определенные элементы матрицы СЛАУ, поэтому можно использовать этот ресурс для ускорения решения СЛАУ. Так, описаны усовершенствования алгоритма вычисления емкостных матриц одной и той же структуры при многократном изменении значения диэлектрической проницаемости [18] и рассмотрено применение алгоритма, использующего блочное LU-разложение, для вычисления емкостных матриц [19, 20]. Эти усовершенствования апробированы в системе TALGAT [21] на практических задачах. Чем меньше изменяющихся элементов, тем быстрее выполняются повторные вычисления.

Однако чаще необходимо моделирование при изменении размеров структуры, что приводит к изменению элементов матрицы, расположенных в произвольных местах [22]. В данном случае алгоритм блочного LU-разложения неприменим. В качестве первого шага в этом направлении было исследовано уменьшение нормы невязки 10-кратного решения систем линейных уравнений,

полученных при малом изменении нескольких параметров структуры [23].

1.3. Решение СЛАУ

1.3.1. Методы решения СЛАУ

Решение СЛАУ вида

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2N}x_N &= b_2 \\ &\vdots \\ a_{N1}x_1 + a_{N2}x_2 + \ldots + a_{NN}x_N &= b_N \end{aligned}\right\}, \qquad (1.14)$$

или в матричной форме

$$\mathbf{Ax} = \mathbf{b}, \qquad (1.15)$$

где N — порядок матрицы; $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ — вектор неизвестных; $\mathbf{b} = (b_1, b_2, \ldots, b_N)$ — вектор свободных членов; $\mathbf{A} = (a_{ij})$ ($i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, N$) — квадратная матрица коэффициентов, заключается в определении вектора неизвестных **x**.

Для анализа матрицы СЛАУ применяют портреты. В случае разреженной матрицы портретом называют множество пар индексов (i, j), таких что $a_{ij} \neq 0$: $P_{\mathbf{A}} = \left\{(i,j)\middle| a_{ij} \neq 0\right\}$ [24]. Для анализа плотных матриц можно использовать несколько множеств, ограниченных значениями, например $P_{\mathbf{A}}(c_1, c_2) = \left\{(i,j)\middle| c_1 < a_{ij} < c_2\right\}$. Также портреты можно представлять графически, присвоив каждому множеству свой цвет.

Методы решения СЛАУ делят на две большие группы. Это прямые (или точные) и итерационные методы [25]. К прямым относятся такие, которые позволяют получить решение за конечное число элементарных операций [26]. Однако они имеют ряд недостатков: накапливание погрешности решения при плохой обусловленности СЛАУ; как правило, вычислительные затраты пропорциональны N^3, что существенно ограничивает их использование. Несмотря на перечисленные недостатки, прямые методы широко

используются на практике [27], что связано с их сравнительной простотой и универсальностью (т.е. они подходят для решения широкого класса задач) [25]. Итерационные методы, в отличие от прямых, основаны на последовательном приближении к решению СЛАУ [28]. С точки зрения вычислительных затрат итерационные методы обычно пропорциональны $N_{it}N^2$ (N_{it} — количество итераций). Из этого следует, что при $N_{it} < N$ (а это часто имеет место) итерационные методы эффективней прямых. Еще одним преимуществом итерационных методов является меньшая погрешность при вычислении плохо обусловленных СЛАУ [25].

При решении СЛАУ необходимо соблюдать корректность постановки задачи [28]. Задача является корректной, если решение существует и единственно. Для подтверждения существования единственного решения СЛАУ необходимо и достаточно неравенства нулю определителя (детерминанта) матрицы **A**. Кроме того, СЛАУ должна быть устойчива к малым возмущениям входных данных [28]. Для оценки устойчивости используют число обусловленности:

$$\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}\| \times \left\|\mathbf{A}^{-1}\right\|, \qquad (1.16)$$

а также «естественное» число обусловленности [29]

$$\nu\delta(\mathbf{x}) = \left\|\mathbf{A}^{-1}\right\| \cdot \|\mathbf{b}\| \big/ \|\mathbf{x}\|.$$

Обычно если СЛАУ плохо обусловлена, то число обусловленности $\mathrm{cond}(\mathbf{A}) >> 1$. На практике вычисление обратной матрицы ($\mathbf{A}^{-1}$) очень трудоемкий процесс, поэтому используют приближенные оценки обусловленности. Один из методов оценки — замена $\left\|\mathbf{A}^{-1}\right\| \approx \max_k \|\mathbf{w}_i\| / \|\mathbf{y}_i\|$, где $\mathbf{y}_i$ — случайный вектор, $i = 1, 2, \ldots, k$; $\mathbf{w}_i$ — решение системы $\mathbf{A}\mathbf{w}_i = \mathbf{y}_i$ [29, 30].

С ростом числа обусловленности и порядка матрицы растет погрешность решения из-за представления чисел с плавающей запятой конечным числом разрядов. Одним из важных следствий этого является невозможность получения корректного решения СЛАУ прямым методом (метод исключения Гаусса) при росте числа обусловленности матриц больших порядков и малой раз-

рядности представления чисел. Поэтому применяют итерационные методы решения СЛАУ, поскольку в них погрешность из-за конечного числа разрядов много меньше, чем в методе Гаусса, так как она не накапливается, а определяется только последней итерацией и не зависит от их числа [13]. Поэтому решение с заданной точностью при росте числа обусловленности матрицы достигается просто увеличением числа итераций. Очевидно, что снижение числа обусловленности матрицы позволяет снизить время решения с заданной точностью за счет уменьшения числа итераций. Кроме того, можно даже уменьшить число разрядов представления чисел с плавающей запятой для снижения времени вычисления и требований к объему памяти компьютера [31].

Следует отметить, что в настоящее время активно развиваются программные средства, позволяющие ускорить решение СЛАУ. Ускорение происходит за счет параллельных вычислений, которые реализуются путем применения многоядерных процессоров и многопроцессорных станций, кластеров, а также графических процессоров. Наиболее популярные программные средства: OpenMP — открытый стандарт для распараллеливания программ [32]; MPI — программный интерфейс для передачи информации, который позволяет обмениваться сообщениями между процессами [33]; CUDA — программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia [34]; GPGPU — техника использования графического процессора видеокарты при выполнении расчетов для общих вычислений [35]; POSIX Threads — стандарт POSIX реализации потоков (нитей) выполнения [36]; Windows API — общее наименование целого набора базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows [37], и др.

Также активно развиваются программные библиотеки для решения СЛАУ. Наиболее популярные программные библиотеки: LAPACK — библиотека с открытым исходным кодом, содержащая методы для решения основных задач линейной алгебры [38]; Automatically Tuned Linear Algebra Software (ATLAS) — программная библиотека для линейной алгебры [39]; Intel Math

Kernel Library (Intel MKL) — библиотека оптимизированных математических подпрограмм [40]; Eigen — библиотека линейной алгебры для C++ с открытым исходным кодом; Basic Linear Algebra Subprograms (BLAS) — стандарт интерфейса программирования приложений для создания библиотек, выполняющих основные операции линейной алгебры [41], программа написана на шаблонах и предназначена для векторно-матричных вычислений и связанных с ними операций [42], и др.

1.3.2. Прямые методы

Исторически первым прямым методом является метод, основанный на исключении неизвестных (метод Гаусса). В настоящее время разработано большое число различных методов, основанных на методе Гаусса [43], далее рассмотрены наиболее распространенные из них.

Метод Гаусса

Это один из самых важных и распространенных прямых методов решения СЛАУ [44], который также называют методом последовательного исключения неизвестных. Он известен в различных вариантах, которые алгебраически тождественны, уже более 2000 лет [28, 45, 46].

Существует множество методов, которые являются модификациями метода Гаусса. Они отличаются порядком исключения неизвестных (этап прямого хода), способами, уменьшающими погрешность вычислений. Например, метод Гаусса – Жордана [44, 47], метод единственного деления [43, 48], с выбором главного элемента [43, 49], метод, основанный на LU-разложении [28, 48, 50], и др. Из перечисленных методов широкое распространение получил метод, основанный на LU-разложении.

Метод LU-разложения

Метод LU-разложения получен из метода Гаусса путем обобщения операций на этапе прямого хода [27, 28]. Метод LU-разложения заключается в представлении матрицы **A** в виде произведения двух матриц, т.е. **A** = **LU**, где **L** — нижняя треугольная матрица; **U** — верхняя треугольная матрица с единицами на глав-

ной диагонали. При выполнении LU-разложения метод Гаусса разбивается на два этапа:

1) вычисление LU-разложения матрицы **A**;

2) вычисление вектора решения [27].

Этот подход позволяет независимо от вектора **b** разложить матрицу **A**, а затем найти решение системы. Если требуется решить несколько систем уравнений с фиксированной матрицей **A** и различными векторами $\mathbf{b}_1$, $\mathbf{b}_2$, …, $\mathbf{b}_p$, то этап LU-разложения проводится один раз. Затем p раз выполняются вычисления второго этапа. Этот подход — главное преимущество перед методом Гаусса, он позволяет снизить число арифметических операций [27, 51].

Существуют разные версии алгоритма метода LU-разложения, которые получены путем изменения порядка обработки элементов [52, 53]. Ниже приведены некоторые из них [50]:

LU-разложение (*ikj*-версия):

```
1   Для i = 2, …, N
2       Для k = 1, …, i – 1
3           a_{i,k} = a_{i,k} / a_{k,k}
4           Для j = k + 1, …, N
5               a_{i,j} = a_{i,j} – a_{i,k}a_{k,j}
6           Увеличить j
7       Увеличить k
8   Увеличить i
```

LU-разложение (*kji*-версия):

```
1   Для k = 1, …, N – 1
2       Для i = k + 1, …, N
3           a_{i,k} = a_{i,k} / a_{k,k}
4       Увеличить i
5       Для j = k + 1, …, N
6           Для i = k + 1, …, N
7               a_{i,j} = a_{i,j} – a_{i,k}a_{k,j}
8           Увеличить i
9       Увеличить j
10  Увеличить k
```

LU-разложение (*jki*-версия):

```
1   Для j = 1, …, N
```

2 Для $k = 1, \ldots, j - 1$
3 Для $i = k + 1, \ldots, N$
4 $a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$
5 Увеличить i
6 Увеличить k
7 Для $i = j + 1, \ldots, N$
8 $a_{i,k} = a_{i,k} / a_{k,k}$
9 Увеличить i
10 Увеличить j

По арифметической сложности эти три версии LU-разложения не отличаются, но очередность важна для компьютерной реализации [53, 54].

Применение LU-разложения возможно не только для решения СЛАУ, но и для следующих важных математических операций:

– определения обратной матрицы;

– нахождения собственных значений матрицы;

– неявного предобусловливания итерационных методов (ILU(0), ILUT и др.) [50, 53].

1.3.3. Итерационные методы

Моделирование все более сложных научно-технических задач в различных областях знаний требует решения СЛАУ все большего порядка [28]. Прямые методы, несмотря на их простоту и универсальность, не позволяют эффективно решать СЛАУ с большими порядками из-за возрастающих вычислительных затрат. Поэтому в последнее время широко применяются итерационные методы. К настоящему времени разработано много различных итерационных методов. Ранние итерационные методы получили название классических, первые из них возникли еще в XIX веке благодаря работам Гаусса – Зейделя [55]. Они основаны на расщеплении исходной матрицы и циклическом изменении вектора решения таким образом, чтобы уменьшить норму вектора невязки ($\mathbf{r} = \mathbf{Ax} - \mathbf{b}$). Эта методика получила название релаксации [24]. Данные методы редко применяются на практике из-за ряда недостатков, например медленной или неустойчивой сходимости и не-

возможности применения предобусловливания [50], и поэтому здесь не рассматриваются. Другое направление развития итерационных методов, используемое и по сей день, основано на подпространствах Крылова. Предпосылкой его стали проекционные методы, возникшие в начале 30-х годов XX века [55].

Проекционные методы

Класс проекционных методов охватывает методы, использующие теорию линейных подпространств. Так, для решения систем вида (1.14) задают K и L — два m-мерных линейных подпространства пространства R^N. Тогда решение сводится к нахождению такого вектора $\tilde{\mathbf{x}} \in K$, чтобы выполнялось условие $\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} \perp L$, т.е. условие $\forall l \in L : (\mathbf{A}\tilde{\mathbf{x}}, l) = (\mathbf{b}, l)$, называемое условием Петрова – Галёркина [24]. Если задать начальное приближение $\mathbf{x}_0$, тогда можно представить $\tilde{\mathbf{x}} = \mathbf{x}_0 + \boldsymbol{\delta}$ и постановку задачи сформулировать следующим образом: найти $\boldsymbol{\delta} \in K$, чтобы выполнялось условие $\forall l \in L : (\mathbf{r}_0 - \mathbf{A}\boldsymbol{\delta}, l) = 0$ [50]. Далее необходимо ввести матричные базисы в подпространствах K и L, например $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m]$ из базисных векторов пространства K и $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m]$ из L. После некоторых преобразований получим

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{V}\left(\mathbf{W}^T\mathbf{A}\mathbf{V}\right)^{-1}\mathbf{W}^T\mathbf{r}_0. \qquad (1.17)$$

Из этого условия сразу вытекает важное требование: подпространства K и L и их базисы выбираются так, чтобы матрица $\mathbf{W}^{\mathrm{T}}\mathbf{A}\mathbf{V}$ либо была малого порядка, либо имела простую структуру, удобную для обращения [24]. Общий вид алгоритма любого проекционного метода:

1 Выбираем пару подпространств K и L
2 Построение для K и L базисов $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m]$ и $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m]$
3 Для $i = 1, 2, \ldots$, пока не достигнута требуемая точность
4 $\quad \mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_{i-1}$
5 $\quad \mathbf{y} = (\mathbf{W}^{\mathrm{T}}\mathbf{A}\mathbf{V})^{-1}\mathbf{W}^{\mathrm{T}}\mathbf{r}_i$
6 $\quad \mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{V}\mathbf{y}$
7 Увеличить i

Выбор пространств K и L будет определять вычислительную схему метода. Наиболее простой случай, когда подпространства K и L одномерны. Тогда соотношение (1.17) принимает вид

$$\mathbf{x}_{k+1} = \mathbf{x}_k + y_k \mathbf{V}_k. \qquad (1.18)$$

Примером являются методы наискорейшего спуска [24], наискорейшего уменьшения невязки [56] и Гаусса – Зейделя [55]. Зачастую эти методы обладают медленной сходимостью [56].

Выделяются методы, для которых в качестве подпространства K выбирается подпространство Крылова. Такие методы называют методами Крыловского типа или методами подпространств Крылова [50]. Эти пространства предложил А.Н. Крылов в 1931 году [57]. Подпространством Крылова размерности m называют линейное пространство

$$K_m(\mathbf{A}, \mathbf{V}) = \operatorname{span}\left\{\mathbf{v}, \mathbf{A}\mathbf{v}, \mathbf{A}^2\mathbf{v}, ..., \mathbf{A}^{m-1}\mathbf{v}\right\}. \qquad (1.19)$$

В качестве вектора $\mathbf{v}$ обычно выбирается невязка начального приближения $\mathbf{r}_0$ ($\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$). Выбор подпространства L и способ построения базисов подпространства определяют вычислительную схему метода [56].

Предобусловливание

Проекционные методы не лишены недостатков. Некоторые методы медленно сходятся [56], у других может происходить стагнация и сбой, что не дает возможности получить решение. Наличие таких недостатков объясняется тем, что скорость сходимости сильно зависит от спектральных свойств матрицы системы, от обусловленности и даже от начального приближения [56]. Для ускорения сходимости применяют методику, называемую предобусловливанием.

Условно методы предобусловливания делят на два вида: явные и неявные [58]. Существуют различные методы предобусловливания, такие как предобусловливание Якоби и Гаусса – Зейделя [56]; неполное LU-разложение [50]; полиноминальное предобусловливание [24] и др.

Явное предобусловливание. Принцип предобусловливания заключается в преобразовании системы $\mathbf{Ax} = \mathbf{b}$ к алгебраически эквивалентной системе [56]:

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}, \tag{1.20}$$

где $\tilde{\mathbf{A}} = \mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$, $\tilde{\mathbf{x}} = \mathbf{M}_2\mathbf{x}$, $\tilde{\mathbf{b}} = \mathbf{M}_1^{-1}\mathbf{b}$, матрицы $\mathbf{M}_1$, $\mathbf{M}_2$ — невырожденные.

Если $\mathbf{M}_1 = \mathbf{E}$ и $\mathbf{M}_2 \neq \mathbf{E}$, то предобусловливание называется правым, если $\mathbf{M}_1 \neq \mathbf{E}$ и $\mathbf{M}_2 = \mathbf{E}$ — левым, а если $\mathbf{M}_1 \neq \mathbf{E}$ и $\mathbf{M}_2 \neq \mathbf{E}$, то предобусловливание называется двухсторонним. Чаще всего используют левое предобусловливание [56], поэтому далее будет рассматриваться только оно. При левом предобусловливании система (1.20) будет выглядеть следующим образом:

$$\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}. \tag{1.21}$$

Основные требования к матрице $\mathbf{M}$: «близость» к матрице $\mathbf{A}$, легкая вычислимость и обратимость [56]. Выбор $\mathbf{M} = \mathbf{A}$ приводит систему $\mathbf{Ax} = \mathbf{b}$ к виду $\mathbf{Ex} = \mathbf{A}^{-1}\mathbf{x}$ ($\mathbf{E}$ — единичная матрица), что дает возможность сразу получить решение, однако это не практично, так как требует нахождения $\mathbf{A}^{-1}$, что является решением системы $\mathbf{Ax} = \mathbf{b}$ [58].

Явное предобусловливание основано на нахождении матрицы $\mathbf{M}^{-1}$. Большое распространение получили методы минимизации функционала. Основная идея таких методов сводится к минимизации

$$\min_{\mathbf{B}\in S} \left\| \mathbf{AB} - \mathbf{E} \right\|_F ,$$

где S — множество разреженных матриц; $\| \cdot \|_F$ — норма Фробениуса, которая находится следующим образом:

$$\left\| \mathbf{AB} - \mathbf{E} \right\|_F^2 = \sum_{j=1}^{n} \left\| \mathbf{Ab}_j - \mathbf{e}_j \right\|_2^2,$$

где $\mathbf{b}_j$ — j-й столбец матрицы $\mathbf{B}$.

Задача нахождения матрицы $\mathbf{B}$ (обратной к матрице $\mathbf{A}$) сводится к решению N независимых линейных задач наименьших квадратов [59].

Такие алгоритмы трудоемки, но имеют высокий потенциал распараллеливания [56, 60]. Самые известные методы минимизации функционала — это SPAI [61], AINV (подробно описан в [62]) и другие [63–65].

Один из первых способов предобусловливания основан на итерационном методе Якоби [50]. В этом случае матрица предобусловливания **M** — диагональная матрица, элементы которой обратны соответствующим диагональным элементам исходной матрицы **A**. Тогда матрица **M** будет состоять только из диагональных элементов, которые равны $1/a_{ii}$, $i = 1, 2, \ldots, N$. На практике предобусловливатель Якоби применятся к матрицам, диагональные элементы которых сильно различаются. В ряде случаев это позволяет уменьшить число обусловленности матрицы, а соответственно и число итераций метода [56].

Неявное предобусловливание. Неявное предобусловливание не требует явного вычисления $\mathbf{M}^{-1}$, но требует решения СЛАУ в каждой итерации [58], при этом система $\mathbf{Ax} = \mathbf{b}$ преобразуется к виду

$$\mathbf{MAx} = \mathbf{Mb}. \qquad (1.22)$$

Большая часть неявных методов основана на представлении $\mathbf{M} = \mathbf{LU}$, где **L** и **U** получены из матрицы $\mathbf{A}_s$ путем LU-разложения. Матрицу $\mathbf{A}_s$ обычно получают из исходной матрицы **A** путем уменьшения количества ненулевых элементов (предфильтрации). Способы предфильтрации достаточно разнообразны, они отличаются подходами к уменьшению количества ненулевых элементов.

При LU-разложении матрицы **L** и **U** получаются более плотными, поэтому часто используют неполное LU-разложение (или ILU-разложение) [56]. Введем S — множество позиций, которые требуется заполнить в произведении **LU**. В общем виде ILU-разложение выглядит следующем образом:

$$a_{ij} \leftarrow \begin{cases} a_{ij} - a_{ik} a_{kk}^{-1} a_{kj}, & a_{ij} \in S \\ a_{ij} & \text{в противном случае} \end{cases} \qquad (1.23)$$

для любого k и $i, j > k$.

Самый простой способ ILU-разложения — это ILU(0), в этом случае множество S совпадает со множеством ненулевых элементов матрицы $\mathbf{A}_s$ [56]. Более сложные способы, такие как ILU(p) [66], ILUT [50] и др., используют анализ значений элементов матрицы $\mathbf{A}_s$ для определения необходимости обнуления [58].

Способы предфильтрации. Исторически предобусловливание применялось к разреженным матрицам, однако Л.Ю. Колотилиной предложен способ, позволивший использовать предобусловливание и для плотных матриц, применив предфильтрацию [67]:

$$a_{ij}^s = 0, \text{ если } \left| a_{ij} \Big/ a^{\max} \right| < \varepsilon, \;\; i,j = 1, \ldots, N, \qquad (1.24)$$

где $a^{\max}$ — максимальный элемент матрицы; ε — порог обнуления.

В данном способе используется нормировка элементов по максимальному элементу в матрице. Таким образом, предфильтрация заключается в получении матрицы $\mathbf{A}_s$ из $\mathbf{A}$ путем обнуления незначительных (малых по величине) элементов. Часто используют сравнение значений элементов матрицы с некоторым порогом (алгебраическая предфильтрация) [58].

В [58] приведен способ, основанный на максимальном элементе в строке:

$$a_{ij}^s = 0, \text{ если } \left| a_{ij} \Big/ a_i^{\max} \right| < \varepsilon, \;\; i,j = 1, \ldots, N,$$

и описаны способы, использующие нормы матриц, например бесконечную:

$$a_{ij}^s = 0, \text{ если } \left| a_{ij} \right| < \varepsilon, \text{ где } \varepsilon = \left\| \mathbf{A} \right\|_\infty \tau, \;\; i,j = 1, \ldots, N;$$

где $\left\| \mathbf{A} \right\|_\infty = \max\limits_j \sum_i a_{ij}$.

Еще один способ предфильтрации реализуется с помощью порога обнуления, получаемого из произведения Фробениусовой нормы матрицы $\mathbf{A}$ и задаваемого допуска обнуления τ [68]:

$$a_{ij}^s = 0, \text{ если } \left| a_{ij} \right| < \varepsilon, \text{ где } \varepsilon = \left\| \mathbf{A} \right\|_F \tau, \;\; i,j = 1, \ldots, N;$$

где $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}|a_{ij}|^2}$.

Методы подпространства Крылова

На основе подпространства Крылова разработано много итерационных методов, которые отличаются размерностью m, способом выбора подпространства L и способом построения базисов [24]. Итерационные методы подпространства Крылова получили широкое распространение в конце XX века и активно развиваются в настоящее время [56]. Такие методы получили название вариационных. В таблице 1.1 приведены наиболее популярные из них [60].

Таблица 1.1 – Наиболее популярные итерационные методы подпространства Крылова

Год	Автор(ы)	Метод	Источник
1950	Lanczos	Lanczos	[70]
1951	Arnoldi	Arnoldi	[71]
1952	Hestenes, Stiefe	CG	[72]
1952	Lanczos	Lanczos (CG)	[73]
1975	Paige, Saunders	MINRES	[74]
1975	Paige, Saunders	SYMMLQ	[75]
1975	Fletcher	BiCG	[76]
1976	Concus, Golub	CGW	[77]
1977	Vinsome	ORTHOMIN	[78]
1977	Meijerink, van der Vorst	ICCG	[79]
1978	Widlund	CGW	[80]
1980	Jea, Young	ORTHODIR	[81]
1981	Saad	FOM	[81]
1982	Paige, Saunders	LSQR	[83]
1983	Eisenstat et al.	GCR	[84]
1986	Saad, Schultz	GMRES	[85]
1989	Sonneveld	CGS	[86]
1991	Freund and Nachtigal	QMR	[87]
1992	van der Vorst	BiCGStab	[88]
1993	Gutknecht	BiCGStab2	[89]
1993	Sleijpen, Fokkema	BiCGStab(l)	[90]
1994	Freund	TFQMR	[91]
1994	Weiss	GMERR	[92]

Окончание таблицы 1.1

Год	Автор(ы)	Метод	Источник
1994	Chan et al.	QMR-BiCGStab	[93]
1995	Kasenally, Ebrahim	GMBACK	[94]
1996	Fokkema et al	CGS2	[95]
1999	de Sturler	GCROT	[96]

Ниже приведены алгоритмы наиболее распространенных итерационных методов.

Алгоритм BiCG

Впервые метод BiCG (метод бисопряженных градиентов) был предложен Ланцошом [73]. В качестве подпространства K выбрано пространство Крылова $K_m(\mathbf{r}_0, \mathbf{A})$, а в качестве пространства L — $L_m\left(\tilde{\mathbf{r}}_0, \mathbf{A}^T\right)$, где $\tilde{\mathbf{r}}_0$ — вектор, при котором $(\mathbf{r}_0, \tilde{\mathbf{r}}_0) \neq 0$. Далее строятся два базиса, ортогональных друг другу. Для построения биортогонального базиса используется процедура Ланцоша [56]. Приближенное решение уточняется по формуле

$$\mathbf{x}_m = \mathbf{x}_0 + \beta \mathbf{V}_m \mathbf{T}_m^{-1} \mathbf{e}_1, \tag{1.25}$$

где $\beta = (\mathbf{r}_0, \tilde{\mathbf{r}}_0)$. Метод требует хранить одновременно все базисные векторы или перевычислять их после нахождения коэффициентов, что неэффективно. Поэтому на практике используют следующий подход.

Запишем LU-разложение матрицы $\mathbf{T}_m$:

$$\mathbf{T}_m = \mathbf{L}_m \mathbf{U}_m$$

и определим матрицу $\mathbf{P}_m$ как

$$\mathbf{P}_m = \mathbf{V}_m \mathbf{U}_m^{-1}. \tag{1.26}$$

Подставим полученное выражение в (1.25):

$$\mathbf{x}_m = \mathbf{x}_0 + \beta \mathbf{V}_m \mathbf{U}_m^{-1} \mathbf{L}_m^{-1} \mathbf{e}_1 = \mathbf{x}_0 + \beta \mathbf{P}_m \mathbf{L}_m^{-1} \mathbf{e}_1$$

и по аналогии с (1.26) запишем

$$\tilde{\mathbf{P}}_m = \mathbf{W}_m \left(\mathbf{L}_m^{\mathrm{T}}\right)^{-1}. \tag{1.27}$$

Тогда можно записать:

$$\tilde{\mathbf{P}}_m^{\mathrm{T}}\mathbf{A}\mathbf{P}_m = \mathbf{L}_m^{-1}\mathbf{W}_m^{\mathrm{T}}\mathbf{A}\mathbf{V}_m\mathbf{U}_m^{-1} = \mathbf{L}_m^{-1}\mathbf{T}_m\mathbf{U}_m^{-1} = \mathbf{D}_m,$$

где $\mathbf{D}_m$ — диагональная матрица с элементами $d_{ii} = (v_i, w_i)$.

Если представить столбцы матрицы $\mathbf{P}_m$ векторами $\mathbf{p}_k$, а столбцы матрицы $\tilde{\mathbf{p}}_m$ — векторами $\tilde{\mathbf{p}}_k$, тогда согласно (1.27) можно записать:

$$i \neq j \rightarrow \tilde{\mathbf{p}}_i^{\mathrm{T}}\mathbf{A}\mathbf{p}_j = \left(\mathbf{A}\mathbf{p}_i, \tilde{\mathbf{p}}_i\right) = 0. \qquad (1.28)$$

Векторы $\left\{\tilde{\mathbf{p}}_i\right\}_{i=1}^m$ и $\left\{\mathbf{p}_i\right\}_{i=1}^m$, удовлетворяющие условию (1.28), называются бисопряженными [56]. Бисопряженность эквивалентна биортогональности скалярного A-произведения, а потому для нахождения векторов $\tilde{\mathbf{p}}_i$ и $\mathbf{p}$ вместо (1.26) и (1.27) можно использовать биортогонализацию Ланцоша [58].

Используя эту информацию, получим алгоритм метода BiCG:

Алгоритм 1.1 — BiCG

1. Выбрать начальное приближение $\mathbf{x}_0$, вычислить $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ и выбрать $\tilde{\mathbf{r}}_0$ так, что $(\mathbf{r}_0, \tilde{\mathbf{r}}_0) \neq 0$ (например, $\tilde{\mathbf{r}}_0 = \mathbf{r}_0$)
2. $\mathbf{p}_0 = \mathbf{r}_0$, $\tilde{\mathbf{p}}_0 = \tilde{\mathbf{r}}_0$
3. Для $j = 1, 2, \ldots$ до сходимости или до $N_{it}^{\max}$
4. Найти $\mathbf{z}_j$ из системы $\mathbf{M}\,\mathbf{z}_j = \mathbf{r}_j$
5. Найти $\tilde{\mathbf{z}}_j$ из системы $\mathbf{M}^{\mathrm{T}}\,\tilde{\mathbf{z}}_j = \tilde{\mathbf{r}}_j$
6. $\alpha_j = (\mathbf{z}_j, \tilde{\mathbf{r}}_j)/(\mathbf{p}_j, \mathbf{A}\tilde{\mathbf{p}}_j)$
7. $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j\mathbf{p}_j$
8. $\mathbf{r}_{j+1} = \mathbf{r}_j + \alpha_j\mathbf{A}\,\mathbf{p}_j$
9. $\tilde{\mathbf{r}}_{j+1} = \tilde{\mathbf{r}}_j + \alpha_j\,\mathbf{A}\tilde{\mathbf{p}}_j$
10. $\beta_j = (\mathbf{z}_{j+1}, \tilde{\mathbf{r}}_{j+1})/(\mathbf{z}_j, \tilde{\mathbf{r}}_j)$
11. $\mathbf{p}_{j+1} = \mathbf{r}_j + \beta_j\mathbf{p}_j$
12. $\tilde{\mathbf{p}}_{j+1} = \tilde{\mathbf{r}}_j + \beta_j\tilde{\mathbf{p}}_j$
13. Увеличить j

Алгоритм BiCGStab

Описанный выше алгоритм BiCG обладает рядом недостатков, среди которых неустойчивость, может возникнуть осциллирующее поведение нормы невязки, итерационный процесс может

полностью оборваться и его нельзя будет продлить [58]. Это происходит, если коэффициент $\beta_j = 0$. Поэтому разработан алгоритм BiCGStab [88]. Далее приведен алгоритм BiCGStab с неявным предобусловливанием.

Алгоритм 1.2 — BiCGStab

1 Выбрать начальное приближение $\mathbf{x}_0$, вычислить $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\,\mathbf{x}_0$
2 Выбрать вектор $\tilde{\mathbf{r}}$, удовлетворяющий условию $(\mathbf{r}_0,\ \tilde{\mathbf{r}}) \neq 0$ (например, $\tilde{\mathbf{r}} = \mathbf{r}_0$)
3 Для $i = 1, 2, \ldots$ до сходимости или до $N_{it}^{\max}$
4 $\rho_{i-1} = (\tilde{\mathbf{r}},\ \mathbf{r}_{i-1})$
5 Если $\rho_{i-1} = 0$
6 то метод не может решить данную систему
7 Если $i = 1$
8 $\mathbf{p}_i = \mathbf{r}_{i-1}$
9 Иначе
10 $\beta_{i-1} = (\rho_{i-1} / \rho_{i-2})\ (\alpha_{i-1} / \omega_{i-1})$
11 $\mathbf{p}_i = \mathbf{r}_{i-1} + \beta_{i-1}(\mathbf{p}_{i-1} - \omega_{i-1}\,\mathbf{v}_{i-1})$
12 Найти $\tilde{\mathbf{p}}$ из системы $\mathbf{M}\,\tilde{\mathbf{p}} = \mathbf{p}_i$
13 $\mathbf{v}_i = \mathbf{A}\,\tilde{\mathbf{p}}$
14 $\alpha_i = \rho_{i-1} / (\tilde{\mathbf{r}},\ \mathbf{v}_i)$
15 $\mathbf{s} = \mathbf{r}_{i-1} - \alpha_i\,\mathbf{v}_i$
16 Если $\|\mathbf{s}\|_2 / \|\mathbf{r}_0\|_2 \le Tol$
17 то КОНЕЦ ($\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\,\tilde{\mathbf{p}}$ – полученное решение)
18 Найти $\tilde{\mathbf{s}}$ из системы $\mathbf{M}\,\tilde{\mathbf{s}} = \mathbf{s}$
19 $\mathbf{t} = \mathbf{A}\,\tilde{\mathbf{s}}$
20 $\omega_i = (\mathbf{t}, \mathbf{s}) / (\mathbf{t}, \mathbf{t})$
21 $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\,\tilde{\mathbf{p}} + \omega_i\,\tilde{\mathbf{s}}$
22 $\mathbf{r}_i = \mathbf{s} - \omega_i\,\mathrm{t}$
23 Если $\|\mathbf{r}\|_2 / \|\mathbf{r}_0\|_2 \le Tol$
24 то КОНЕЦ ($\mathbf{x}_i$ – полученное решение)
25 Увеличить i

Алгоритм CGS

Основная идея алгоритма CGS заключается в следующем: невязку $\mathbf{r}_j$ можно выразить как полином от начальной невязки $\mathbf{r}_j = \mathbf{p}_j(\mathbf{A})\mathbf{r}_0$. Аналогично полином сопряженного пространства

$\tilde{\mathbf{r}}_j = \mathbf{p}_j\left(\mathbf{A}^{\mathrm{T}}\right)\tilde{\mathbf{r}}_0$. Для BiCG скалярное произведение $(\mathbf{r}_j,\ \tilde{\mathbf{r}}_j)$ примет вид $\left(\mathbf{p}_j(\mathbf{A})\mathbf{r}_0, \mathbf{p}_j\left(\mathbf{A}^{\mathrm{T}}\right)\tilde{\mathbf{r}}_0\right) \equiv \left(\mathbf{p}_2^j(\mathbf{A})\mathbf{r}_0, \tilde{\mathbf{r}}_0\right)$. Это наблюдение привело к соотношению $\tilde{\mathbf{x}}_j \in K^{2j}(\mathbf{A}, \mathbf{r}_0)$, для которого можно записать невязку $\tilde{\mathbf{r}}_j = p_2^j(\mathbf{A})\mathbf{r}_0$. Следовательно, невязка уменьшается в два раза быстрее [56]. Далее приведен алгоритм метода CGS.

Алгоритм 1.3 — CGS

1 Выбрать начальное приближение $\mathbf{x}_0$, вычислить $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\,\mathbf{x}_0$
2 Выбрать вектор $\tilde{\mathbf{r}}$, удовлетворяющий условию $(\mathbf{r}_0,\ \tilde{\mathbf{r}}) \neq 0$ (например, $\tilde{\mathbf{r}} = \mathbf{r}_0$)
3 Для $i = 1, 2, \ldots$ до сходимости или до $N_{it}^{\max}$
4 $\rho_{i-1} = (\tilde{\mathbf{r}},\ \mathbf{r}_{i-1})$
5 Если $\rho_{i-1} = 0$
6 то метод не может решить данную систему
7 Если $i = 1$
8 $\mathbf{u}_1 = \mathbf{r}_0$
9 $\mathbf{p}_1 = \mathbf{u}_1$
10 Иначе
11 $\beta_{i-1} = (\rho_{i-1} / \rho_{i-2})$
12 $\mathbf{u}_i = \mathbf{r}_i + \beta_{i-1}\mathbf{q}_{i-1}$
13 $\mathbf{p}_i = \mathbf{u}_i + \beta_{i-1}(\mathbf{q}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1})$
14 Найти $\tilde{\mathbf{p}}$ из системы $\mathbf{M}\tilde{\mathbf{p}} = \mathbf{p}_i$
15 $\tilde{\mathbf{v}} = \mathbf{A}\tilde{\mathbf{p}}$
16 $\alpha_i = \rho_{i-1} / (\tilde{\mathbf{r}}, \tilde{\mathbf{v}})$
17 $\mathbf{q}_i = \mathbf{u}_i - \alpha_i\,\tilde{\mathbf{v}}$
18 Найти $\tilde{\mathbf{u}}$ из системы $\mathbf{M}\tilde{\mathbf{u}} = \mathbf{u}_i + \mathbf{q}_i$
19 $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\tilde{\mathbf{u}}$
20 $\tilde{\mathbf{q}} = \mathbf{A}\tilde{\mathbf{u}}$
21 $\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i\,\tilde{\mathbf{q}}$
22 Если $\|\mathbf{r}\|_2 / \|\mathbf{r}_0\|_2 \le Tol$
23 то КОНЕЦ ($\mathbf{x}_i$ – полученное решение)
24 Увеличить i

1.4. Хранение разреженных матриц

1.4.1. Определение разреженной матрицы

Для уменьшения вычислительных затрат в итерационных методах используют предобусловливание. Однако это требует хранения дополнительной матрицы, что увеличивает объем памяти компьютера. Особенность дополнительной матрицы в том, что она, как правило, является разреженной. Хранить ее можно с помощью любого из форматов хранения разреженных матриц для экономии машинной памяти.

Разреженная матрица — матрица, имеющая преимущественно нулевые элементы. Объем ненулевых элементов в общем случае определяется для каждой отдельной задачи [97]. Одно из определений принадлежит Альварадо (1979): матрица порядка N, число ее ненулевых элементов должно выражаться как $N^{1+\gamma}$, где $\gamma < 1$. Матрица разрежена, если $\gamma \le 0{,}2$, и для ленточных матриц, если $\gamma \le 0{,}5$ [98].

Другое определение заключается в том, что матрицу следует считать разреженной, если удается извлечь выгоду из наличия в ней нулей [97]. Такой выгодой может стать, например, уменьшение памяти компьютера для хранения матрицы в сжатом формате, в отличие от формата, не учитывающего наличие нулевых элементов.

1.4.2. Форматы хранения разреженных матриц

Существует много различных форматов хранения разреженных матриц [97, 99]. 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.) [99]. Как правило, форматы, обеспечивающие хорошее сжатие, требуют для операций больше машинного времени и наоборот [97, 99]. Поэтому необходимо выбирать формат для каждой конкретной задачи.

Из наиболее известных следует выделить форматы из двух векторов [99], Кнута [100], Рейнболдта и Местеньи [101], Ларкума

[102], Шермана [103], разреженный строчный (Compressed Storage Row, CSR), разреженный столбцовый (Compressed Storage Column, CSC) [104, 105], разреженный неравномерный диагональный (Jagged Diagonal Storage, JDS) [106], сжатый диагональный (Compressed Diagonal Storage, CDS) [107], разреженный блочный строчный (Sparse Block Compressed Row Storage, SBCRS) [108] и др. [109–112].

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

Рассмотрим самые распространенные форматы хранения разреженных матриц.

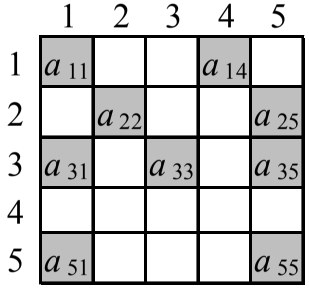
Формат из двух векторов

При использовании формата из двух векторов матрица хранится в виде двух векторов — **I** и **A** [99]. Вектор **I** содержит номер столбца, **A** — значения ненулевых элементов. Если элемент в векторе **I** равен нулю, это означает начало строки, при этом элемент вектора **A** содержит номер следующей строки. Если нули в двух векторах, это означает конец матрицы. На рисунке 1.3 приведен пример формата.

Основным преимуществом формата является то, что он обеспечивает хороший коэффициент сжатия [99], недостатком — сложность выполнения элементарных операций, таких как поиск и добавление элемента матрицы.

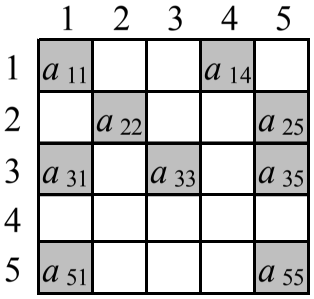
Формат Кнута

Формат предложен Д.Э. Кнутом в 1968 г. [100]. 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 используются дополнительные четыре вектора, в которых хранятся указатели на позиции ненулевого элемента в векторе **AN**.



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 1.3 – Представление матрицы с помощью формата из двух векторов



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

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 1.4 приведен пример представления матрицы с помощью формата Кнута.

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 [97]. Достоинством формата является легкость добавления (или удаления) элементов матрицы, поэтому он хорошо подходит для алгоритмов, где нельзя предсказать конечное число ненулевых элементов [97] (например, LU-разложение).

Формат Рейнболдта и Местеньи

Формат Рейнболдта и Местеньи (1973) является модификаций формата Кнута [101]. В отличие от формата Кнута, векторы **NR** и **NC** закольцовываются, т.е. последний элемент в строке (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), так как заранее неизвестны индексы ее ненулевых элементов. Пример приведен на рисунке 1.5.

Формат Ларкума

Усовершенствованный вариант формата Кнута использовался Ларкумом (1971) для хранения симметричных (или треугольных) матриц (рисунок 1.6) [102].

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 можно хранить несимметричные матрицы.

	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				

Рисунок1.5 – Представление матрицы с помощью формата Рейнболдта и Местеньи
(** – в данной строке нет ненулевых элементов)

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

Формат Шермана

Еще один из форматов (рисунок 1.7) для треугольных матриц был предложен Шерманом (1975) [103]. 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 1.8), **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 1.7 – Представление матрицы с помощью формата Шермана (* – в данной строке нет элементов; ** – правило сжатия представлено на рисунке 1.8)

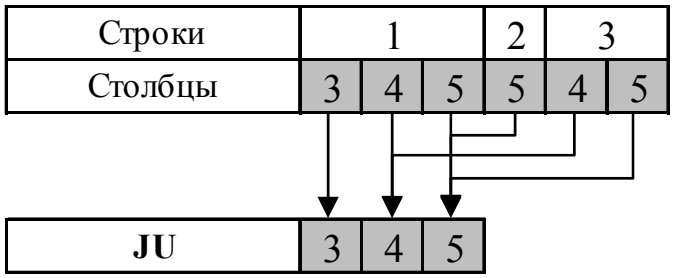


Рисунок 1.8 – Сжатие вектора JU

Формат CSR (CSC)

Форматы CSR и CSC, предложенные Чангом (1969) и Густавсоном (1972), широко используются для хранения разреженных матриц [104, 105]. 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. [50]. В формате CSR значения ненулевых элементов матрицы и соответствующие им индексы столбцов хранятся по строкам в двух векторах (**Values** — ненулевые элементы, **Columns** — индексы столбцов). Используется также вектор указателей на ненулевые элементы (**RowIndex**), с которых начинается очередная строка (рисунок 1.9). Формат CSC отличается только порядком хранения ненулевых элементов матрицы, так, в CSR элементы хранятся по строкам, в CSC — по столбцам, при этом используются векторы **Values**, **Rows**, **ColumnsIndex** соответственно (рисунок 1.10).

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 матриц **Val** и **Col** хранятся нули (рисунок 1.11).

Однако при таком способе хранения возникает избыточность, которая устраняется дополнительным сжатием матрицы **Val** по столбцам. В результате получаются три вектора (рисунок 1.12): **Values** (значения ненулевых элементов), **Columns** (индекс столбца ненулевого элемента), **StartPosition** (указатель на начальный элемент в столбце матрицы **Val**).

Сжатый диагональный формат хранения (CDS)

Сжатый диагональный формат (CDS) описан в [97, 99]. Этот формат предназначен для хранения ленточных матриц (матрицы, у которых ненулевые элементы расположены только на главной диагонали и примыкающих к ней [50]). Исходная матрица представляется матрицей **Val**, где диагонали исходной матрицы хранятся по строкам, начиная слева направо (рисунок 1.13). При этом нет необходимости хранить дополнительную информацию [97].

	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 1.9 – 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 1.10 – Представление матрицы с помощью формата CSC

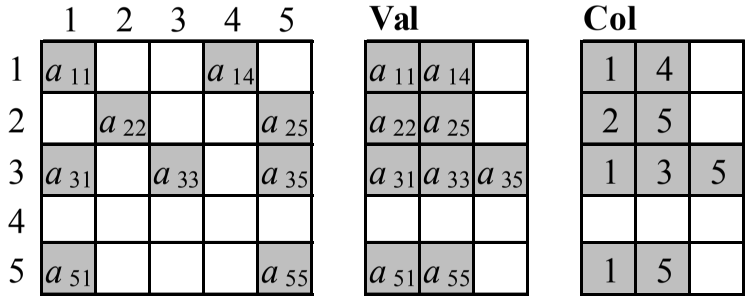


Рисунок 1.11 – Первый шаг сжатия матрицы с помощью формата JDS

Val

a_{11}	a_{14}	
a_{22}	a_{25}	
a_{31}	a_{33}	a_{35}
a_{51}	a_{55}	

Col

1	4	
2	5	
1	3	5
1	5	

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 1.12 – Представление матрицы с помощью формата JDS

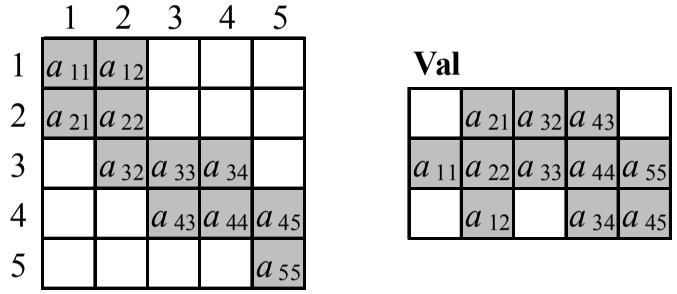


Рисунок 1.13 – Представление матрицы с помощью формата CDS

1.5. Обзор методов многократного решения СЛАУ

Существуют затратные по времени задачи, в которых необходимо многократное решение СЛАУ. В случае когда изменяются все составляющие СЛАУ, задача многократного решения сводится к уравнению

$$\mathbf{A}_k \mathbf{x}_k = \mathbf{b}_k,\ k = 1, 2, ..., m, \tag{1.29}$$

где k — порядковый номер СЛАУ; m — количество СЛАУ.

Необходимость решения уравнения вида (1.29) возникает во многих задачах научных и инженерных вычислений [113], например в методах, использующих рекурсивные вычисления наименьших квадратов [114, 115], задачах рассеяния волн [116], реставрации изображений [117] и др. Рассмотрим несколько подходов к решению уравнения вида (1.29) [118].

Частным случаем многократного решения СЛАУ являются задачи, в которых неизменной остается одна из составляющих СЛАУ, например при неизменной матрице **A** задача может быть записана в виде [119]

$$\mathbf{AX} = \mathbf{B}, \tag{1.30}$$

где $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m]$ и $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_m]$.

В этом случае задача упрощается и для ее решения могут быть использованы итерационные методы. Например, описана методика использования подпространства Крылова для такой задачи. При этом подпространство будет выглядеть следующим образом:

$$K_m(\mathbf{A}, \mathbf{R}) = \text{span}\left\{\mathbf{R}, \mathbf{AR}, \mathbf{A}^2\mathbf{R}, ..., \mathbf{A}^{m-1}\mathbf{R}\right\},$$

где **R** — блок векторов $N \times m$, а матричные базисы подпространств K и L представляются в виде

$$\mathbf{V} = \left[\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_m\right],$$

где **V** — блок векторов $N \times m$;

$$\mathbf{W} = \left[\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_m\right],$$

где **W** — блок векторов $N \times m$ [113, 120].

Таким образом, практически любой проекционный итерационный метод может быть адаптирован для решения подобных задач. Например, разработаны новые итерационные методы GL-LSQR [120], MHGMRES(m) [121], MEGCR [122] и др.

Еще одним частным случаем являются задачи, в которых на каждом шаге меняется матрица **A**, а вектор **b** неизменен. Пример такой задачи — вычисление емкостных матриц полосковых структур в диапазоне изменения их параметров. Так как вектор **b** неизменен, то

$$\mathbf{A}_k \mathbf{x}_k = \mathbf{b},\ k = 1, 2, ..., m. \qquad (1.31)$$

Представим задачу вычисления емкостных матриц, полученных из одной и той же структуры при изменении значения диэлектрической проницаемости [18, 123]. В этом случае изменяются элементы, находящиеся только на главной диагонали нижней правой подматрицы (с индексами элементов матрицы СЛАУ больше N_c), соответствующие подынтервалам диэлектрик-диэлектрик (см. рисунок 1.2), поэтому можно использовать алгоритмы, основанные на блочном LU-разложении [18, 123–125]. В упомянутых работах показана возможность существенного снижения временных затрат.

Однако при изменении размеров структуры, что приводит к изменению элементов матрицы, расположенных в произвольных местах [126], использование блочного LU-разложения невозможно. Такие задачи могут решаться итерационными методами с предобусловливанием, причем предобусловливатель, полученный для одной матрицы, можно использовать и для других. В качестве первого шага в этом направлении было исследовано уменьшение нормы невязки 10-кратного решения СЛАУ, полученных при малом изменении нескольких параметров структуры [23]. Недостатком данного подхода является то, что переформирование предобусловливателя — это трудоемкий процесс, и при сильных изменениях в матрице будет требоваться постоянное переформирование, что неэффективно. Однако существуют методы, использующие корректировку предобусловливателя [118]. В случае неявного предобусловливания корректировка матриц **L** и **U** достаточно сложна и почти всегда неэффективна [118]. В случае яв-

ного предобусловливания существуют методы пошагового приближения к обратной матрице, причем эту процедуру можно повторять на каждом шаге. Однако эти методы отличаются большой трудоемкостью, особенно для плотных матриц.

Таким образом, обзор показал актуальность уменьшения времени имитационного моделирования. При этом большая часть времени приходится на решение СЛАУ, следовательно, уменьшая время решения СЛАУ, можно сократить время имитационного моделирования в целом. Обзор методов решения СЛАУ выявил перспективность использования итерационных методов подпространств Крылова с применением предобусловливателя. При предобусловливании используют разреженную матрицу, полученную из исходной путем предфильтрации. Для хранения разреженной матрицы служат специальные форматы, что позволяет добиться экономии машинной памяти и в перспективе — времени решения СЛАУ.

На практике часто требуется вычисление емкостной матрицы при изменении параметров структуры, что очень затратно по времени, так как приходится многократно решать СЛАУ. Из обзора следует, что для решения таких задач можно использовать итерационные методы с предобусловливанием и в перспективе снизить временные затраты.

Цель работы — усовершенствовать алгоритмы решения СЛАУ в задачах моделирования ЭМС.

Для достижения этой цели необходимо решить следующие задачи:

1) разработать новые алгоритмы итерационных методов решения СЛАУ с предобусловливанием, использующие форматы хранения разреженных матриц;

2) разработать новые алгоритмы многократного решения СЛАУ итерационными методами с предобусловливанием;

3) апробировать полученные алгоритмы и оценить их эффективность;

4) разработать комплекс программ для реализации новых алгоритмов.

2. Ускорение решения СЛАУ с помощью форматов хранения разреженных матриц

2.1. Аналитические оценки сжатия данных

Для формата хранения разреженных матриц важным показателем является коэффициент сжатия *k*, который он может обеспечить [127]. Коэффициент сжатия определяется как отношение N^2 (N — порядок матрицы) к числу хранимых данных в разреженном формате. Для определения коэффициента сжатия можно использовать подсчет объема данных конкретной матрицы, хранящихся в сжатом формате, но удобнее использовать аналитические выражения, позволяющие получить простую оценку числа данных. Аналитические выражения определены исходя из структуры конкретного формата хранения. При этом тип данных не учитывается (хранятся как сами ненулевые элементы, так и указатели, для которых требуется меньше машинной памяти). В результате получена нижняя граница коэффициента сжатия, который зависит от порядка матрицы N и количества ненулевых элементов $N_{\text{н}}$ в ней. Исключением в рассматриваемых форматах являются формат Шермана, который зависит от портрета ненулевых элементов (поэтому здесь приведены значения для оптимального, обеспечивающего максимальное сжатие, и неоптимального портретов матрицы); формат CDS, который зависит от числа диагоналей, содержащих ненулевые элементы (N_c). В таблице 2.1 представлены полученные выражения для рассматриваемых форматов хранения разреженных матриц. Видно, что некоторые форматы при одинаковом числе ненулевых элементов $N_{\text{н}}$ имеют одинаковый коэффициент сжатия, например формат двух векторов CSR и JDS.

Оценка коэффициента сжатия с помощью числа ненулевых элементов не всегда удобна. Это связано с тем, что число ненулевых элементов не может однозначно определить коэффициент сжатия формата, так как при одинаковом числе ненулевых элементов, но при разном порядке коэффициент сжатия будет разный. Чтобы избежать неоднозначности, предложено использовать

оценку числа ненулевых элементов в матрице с помощью показателя плотности матрицы q, представляющего собой отношение числа ее ненулевых элементов к N^2.

Таблица 2.1 – Коэффициент сжатия для разных форматов

Формат	Коэффициент сжатия
Два вектора	$k = N^2/(2N_н + N)$
Кнута	$k = N^2/(5N_н + 2N)$
Рейнболдта и Местеньи	$k = N^2/(3N_н + 2N)$
Ларкума*	$k = N^2/(3N_н)$
Шермана*	$k^{***}_{min} = N^2/(3N_н),\ k^{***}_{max} = N^2/(2N_н)$
CSR	$k = N^2/(2N_н + N)$
JDS	$k = N^2/(2N_н + N)$
CDS**	$k = N^2/(N_c N)$
* — только для треугольных матриц; ** — только для ленточных матриц; *** — для оптимального (k_{max}) и неоптимального (k_{min}) портретов матрицы; $N_н$ — число ненулевых элементов матрицы; N_c — число диагоналей, содержащих ненулевые элементы	

Для примера на рисунке 2.1 представлены рассчитанные по полученным формулам (см. таблицу 2.1) зависимости коэффициента сжатия форматов CSR и Кнута от плотности матрицы при $N = 1000$.

Видно, что формат хранения разреженной матрицы не всегда эффективен, т.е. он может не уменьшать, а увеличивать число хранимых данных (все значения q на рисунке 2.1, для которых $k < 1$). Очевидно, что для каждого формата существует граница эффективности (значение q, при котором $k = 1$), значение которой можно оценить аналитически. В таблице 2.2 приведены формулы для оценки плотности матрицы, ниже которой использование форматов неэффективно.

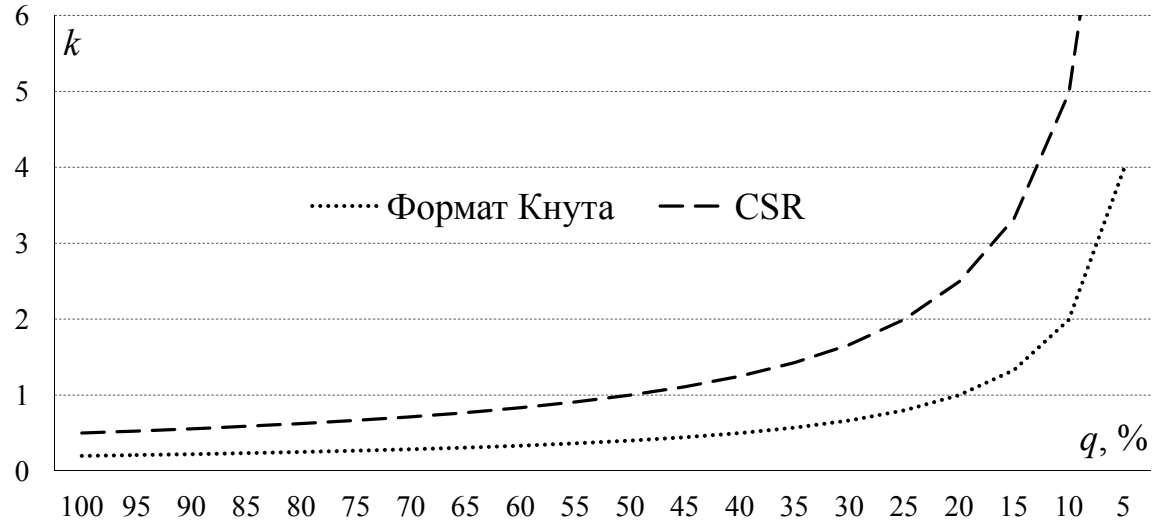


Рисунок 2.1 – Зависимости коэффициентов сжатия от плотности матрицы для форматов Кнута и CSR при $N = 1000$

Таблица 2.2 – Граница эффективности форматов хранения разреженных матриц

Формат	Граница эффективности формата
Два вектора	$q = 100\,\%\left(N^2 - N\right)\Big/\left(2N^2\right)$
Кнута	$q = 100\,\%\left(N^2 - 2N\right)\Big/\left(5N^2\right)$
Рейнболдта и Местеньи	$q = 100\,\%\left(N^2 - 2N\right)\Big/\left(3N^2\right)$
Ларкума*	$q = 100\,\%\left(N^2\right)\Big/\left(3N^2\right)$
Шермана*	$q^{***}_{\min} = 100\,\%\left(N^2\right)\Big/\left(3N^2\right)$ $q^{***}_{\max} = 100\,\%\left(N^2\right)\Big/\left(2N^2\right)$
CSR	$q = 100\,\%\left(N^2 - N\right)\Big/\left(2N^2\right)$
JDS	$q = 100\,\%\left(N^2 - N\right)\Big/\left(2N^2\right)$
CDS**	$q = 100\,\%\left(N^2\right)\Big/\left(N_c N^2\right)$
* — только для треугольных матриц; ** — только для ленточных матриц; *** — для оптимального ($q_{\min}$) и неоптимального ($q_{\max}$) портретов матрицы; N_c — число диагоналей, содержащих ненулевые элементы	

Оценим зависимость коэффициента сжатия от порядка матрицы. На рисунке 2.2 показаны зависимости k от N при $q = 10\ \%$ для разных форматов. Видно, что коэффициент сжатия с ростом N стремится к максимально возможному значению. Предел выражения для k при $N \to \infty$ дает аналитическую оценку для определения максимального значения k (таблица 2.3).

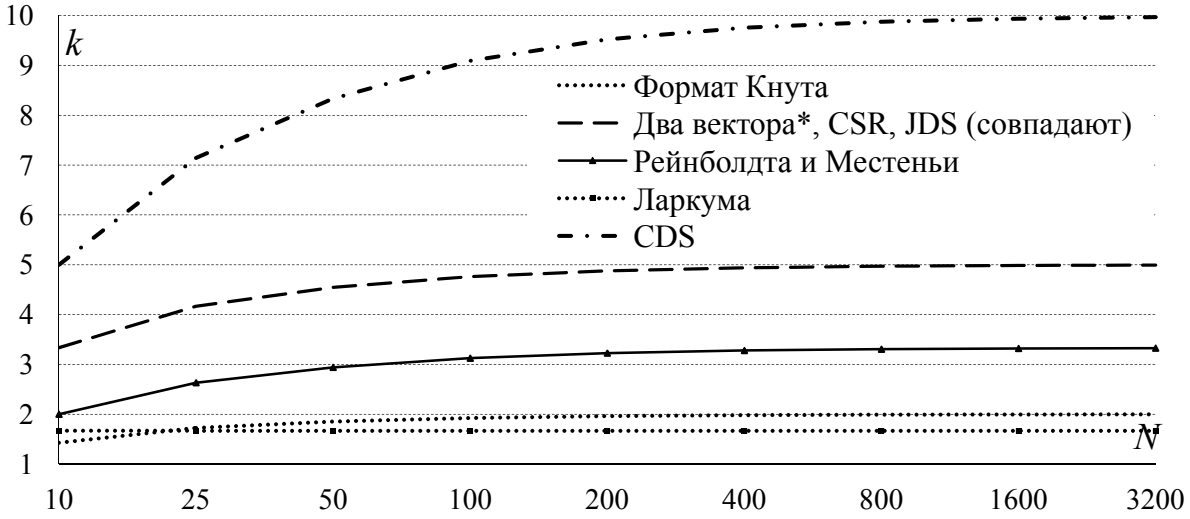


Рисунок 2.2 – Зависимости коэффициентов сжатия k от N при $q = 10\ \%$ для разных форматов

Из рисунка 2.2 видно, что формат CDS обеспечивает максимальное сжатие, однако он подходит только для ленточных матриц и не пригоден для хранения других видов матриц.

Из таблицы 2.3 следует, что максимальный коэффициент сжатия обеспечивают форматы двух векторов, CSR и JDS. Однако при выборе формата хранения разреженной матрицы необходимо учитывать не только эффективность сжатия формата, но и применимость его в алгоритмах. Поэтому для дальнейшего использования выбран формат CSR, так как он наиболее удобен для многих важных операций над разреженными матрицами, таких как сложение, умножение, перестановка строк и столбцов, транспонирование, решение СЛАУ с разреженными матрицами прямыми и итерационными методами и т.д. [24, 50, 97].

Таблица 2.3 – Максимальный коэффициент сжатия для разных форматов

Формат	Максимальный коэффициент сжатия
Два вектора	$k = 1/(2q)$
Кнута	$k = 1/(5q)$
Рейнболдта и Местеньи	$k = 1/(3q)$
Ларкума*	$k = 1/(3q)$
Шермана*	$k^{***} = 1/(2q)$
CSR	$k = 1/(2q)$
JDS	$k = 1/(2q)$
CDS**	$k = 1/(N_c q)$
* — только для треугольных матриц; ** — только для ленточных матриц; *** — для оптимального портрета матрицы; q — плотность матрицы; N_c — число диагоналей, содержащих ненулевые элементы	

2.2. Применение формата CSR для ускорения решения СЛАУ

2.2.1. ILU(0)-разложение

Для реализации ILU(0)-разложения выбран алгоритм *ikj*-версии, так как в этом алгоритме элементы матриц **L** и **U** вычисляются по строкам, вследствие чего наиболее выгодно использовать формат CSR. Приведем алгоритм ILU(0)-разложения [106].

Алгоритм 2.1 – ILU(0)-разложение (*ikj*-версия)

1 Для $i = 2, \ldots, N$
2 Для $k = 1, \ldots, i - 1$
3 Если $a_{i,k} \neq 0$
4 $a_{i,k} = a_{i,k} / a_{k,k}$
5 Для $j = k + 1, \ldots, N$
6 Если $a_{i,j} \neq 0$
7 $a_{i,j} = a_{i,j} - a_{i,k} a_{k,j}$
8 Увеличить j
9 Увеличить k
10 Увеличить i

Данный алгоритм позволяет пересчитать i-ю строку матрицы **A** в i-ю строку матриц **L** и **U**. Первые j–1 строк матрицы **A** участвуют только в определении j-й строки матриц **L** и **U**, что позволяет при вычислениях не хранить их. В формате CSR получить прямой доступ к ненулевому элементу невозможно, так как не хранится прямой адрес элемента. Самый простой вариант доступа к элементу — использование функции его поиска. Но при этом алгоритм будет очень медленным. В результате тестирования оказалось, что алгоритм, использующий разреженный формат хранения, работает примерно в 500 раз дольше (для $N = 1000$) обычного алгоритма без сжатия матрицы [127]. Следовательно, алгоритм в таком виде использовать нельзя, его необходимо усовершенствовать.

Если отметить элементы матрицы, используемые в цикле по j (строки 5–8) алгоритма 2.1, то получится картина, изображенная на рисунке 2.3. Как видно, расчет производится с использованием строк i и k. Поэтому здесь возможно использование цикла по ненулевым элементам в формате CSR. Тогда поиск элементов не нужен, операция (строка 7 в алгоритме 2.1) будет выполняться только с элементами строк i и k, у которых равны индексы столбцов. В результате разработан алгоритм 2.2.

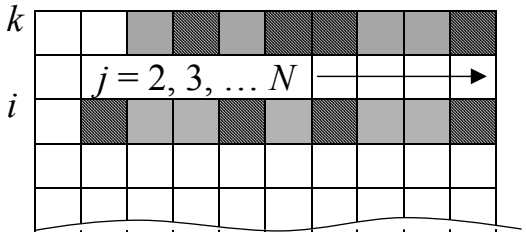


Рисунок 2.3 – Схематичное расположение элементов, используемых за один цикл алгоритма ILU(0)-разложения (*ikj*-версия). Штриховкой отмечены ненулевые элементы, серым – нулевые, белым – элементы, не используемые в цикле

Алгоритм 2.2 – ILU(0)-разложение с использованием формата CSR

1 Для $i = 2, \ldots, N$
2 Для $k = 1, \ldots, i - 1$
3 Найти s – номер элемента $a^s_{i,k}$ в векторе **aelem**

```
4               Если aelem(s) ≠ 0
5                   aelem(s) = aelem(s) / Poisk(k,k)
6               Найти T – номер элемента a^s_{i,k} в векторе aelem
7               Elem1 = T + 1
8               Найти T – номер элемента a^s_{i,k+1} в векторе aelem
9               Elem2 = T
10              pr = Истина
11              Пока pr = Истина
12                  Если jptr(Elem1) = jptr(Elem2)
13                      aelem(Elem1) = aelem(Elem1) – aelem(T) ×
                            aelem(Elem2)
14                      Elem1= Elem1 + 1
                        Elem2 = Elem2 + 1
15                  Если jptr(Elem1) > jptr(Elem2)
16                      Elem2 = Elem2 + 1
17                  Если jptr(Elem1) < jptr(Elem2)
18                      Elem1 = Elem1 + 1
19                  Если iptr(k+1) < Elem2 или iptr(i+1) < Elem2
20                      pr = Ложь
21          Увеличить k
22  Увеличить i
```

В алгоритме 2.2 происходит поиск (строка 3) элемента $a^s_{i,k}$, который используется в алгоритме 2.1 в строках 3, 4 и 7, поэтому позиция элемента запоминается (переменная *s*). В строке 5 присутствует функция *Poisk*(*k*, *k*) (алгоритм 2.3), которая предназначена для поиска элемента $a^s_{k,k}$ в матрице, который используется для выполнения операции в строке 4 алгоритма 2.1.

Алгоритм 2.3 – **Функция поиска** *Poisk*(*k*, *k*)

```
1   Vhod1 = iptr[k]
2   Vhod2 = iptr[k+1]
3   Для i = Vhod1, …, Vhod2
4           Если jptr[i]=k
5                   Возврат aelem[i]
6   Увеличить i
```

Переменная *pr* сигнализирует, когда достигнут конец строки *i* или *k* в матрице и требуется прервать цикл. Переменные *Elem1*

и *Elem2* определяют текущий элемент в первой (*i*-й) и второй (*k*-й) строках соответственно. Переход по элементам строк в матрице осуществляется в цикле по следующему принципу: сравниваются индексы элементов двух текущих строк, индекс с меньшим значением инкрементируется (строки 15–18), если индексы элементов равны (т.е. существуют ненулевые элементы в двух строках текущего столбца), то выполняется операция и значения индексов обеих переменных инкрементируются (строки 12–14).

Анализ показал, что поиск диагонального элемента в строке 5 алгоритма 2.2 существенно увеличивает время его работы [128]. Поэтому целесообразно ввести в формат хранения дополнительный вектор **Diag**, в котором будут храниться указатели на диагональные элементы. С использованием вектора **Diag** разработан алгоритм 2.4.

Алгоритм 2.4 обладает недостатком: тратится время на поиски ненулевых элементов матрицы в строках 3, 6, 8. Данные поиски необходимы, так как при использовании формата CSR нет прямой ссылки на первый ненулевой элемент в цикле *j* (строка 5 в алгоритме 2.1). Таким образом, если избавиться от поиска ненулевых элементов, то можно сократить затрачиваемое машинное время.

Обратимся к исходному алгоритму 2.1 и расположим схематично элементы матрицы, используемые в алгоритме, при $N = 4$ (рисунок 2.4). Изображения матрицы поделены по шагам так, что один шаг соответствует одному значению переменных *i* и *k*. Видно, что элементы a_{ik} участвуют в цикле, который всегда начинается с первого элемента строки, и перемещаются по строкам до последнего элемента в нижнетреугольной матрице. Данную цикличность можно использовать в алгоритме с форматом CSR, поскольку движение элемента происходит по строкам. Причем в этом случае нет необходимости искать элемент a_{ik} в каждом цикле (строка 3 в алгоритме 2.4).

В алгоритме 2.4 имеется еще одна особенность: начальный элемент цикла *kj* всегда находится после соответствующего диагонального элемента. Этот факт позволяет исключить поиск элемента a_{kj} (строка 8 алгоритма 2.4).

Алгоритм 2.4 – ILU(0)-разложение с использованием формата CSR с дополнительным вектором **Diag**

```
1       Для i = 2, …, N
2           Для k = 1, …, i – 1
3               Найти s – номер элемента a^s_{i,k} в векторе aelem
4               Если aelem(s) ≠ 0
5                   aelem(s) = aelem(s) / aelem(Diag(k))
6               Найти T – номер элемента a^s_{i,k} в векторе aelem
7               Elem1 = T + 1
8               Найти T – номер элемента a^s_{i,k+1} в векторе aelem
9               Elem2 = T
10              pr = Истина
11              Пока pr = Истина
12                  Если jptr(Elem1) = jptr(Elem2)
13                      aelem(Elem1) = aelem(Elem1) –
                            aelem(T) × aelem(Elem2)
14                      Elem1 = Elem1 + 1
                        Elem2 = Elem2 + 1
15                  Если jptr(Elem1) > jptr(Elem2)
16                      Elem2 = Elem2 + 1
17                  Если jptr(Elem1) < jptr(Elem2)
18                      Elem1 = Elem1 + 1
19                  Если iptr(k+1) < Elem2 или iptr(i+1) < Elem2
20                      pr = Ложь
21          Увеличить k
22      Увеличить i
```

С учетом этих особенностей поиски ненулевых элементов в алгоритме 2.4 можно исключить. В результате разработан алгоритм 2.5.

Видно, что в алгоритме 2.5 существует избыточность: большое количество сравнений в строках 19–27. В этих строках происходит сравнение индексов столбцов ненулевых элементов, находящихся в одной строке, с индексами ненулевых элементов другой строки, и если совпадение обнаружено, то выполняется операция над этими элементами. Схематично данная операция изображена на рисунке 2.5 (результат выполнения операций над элементами строк обозначен как Вектор-результат). Для выполнения этой операции необходимо перебирать и сравнивать между

собой все индексы столбцов ненулевых элементов двух векторов (строк), что неэффективно.

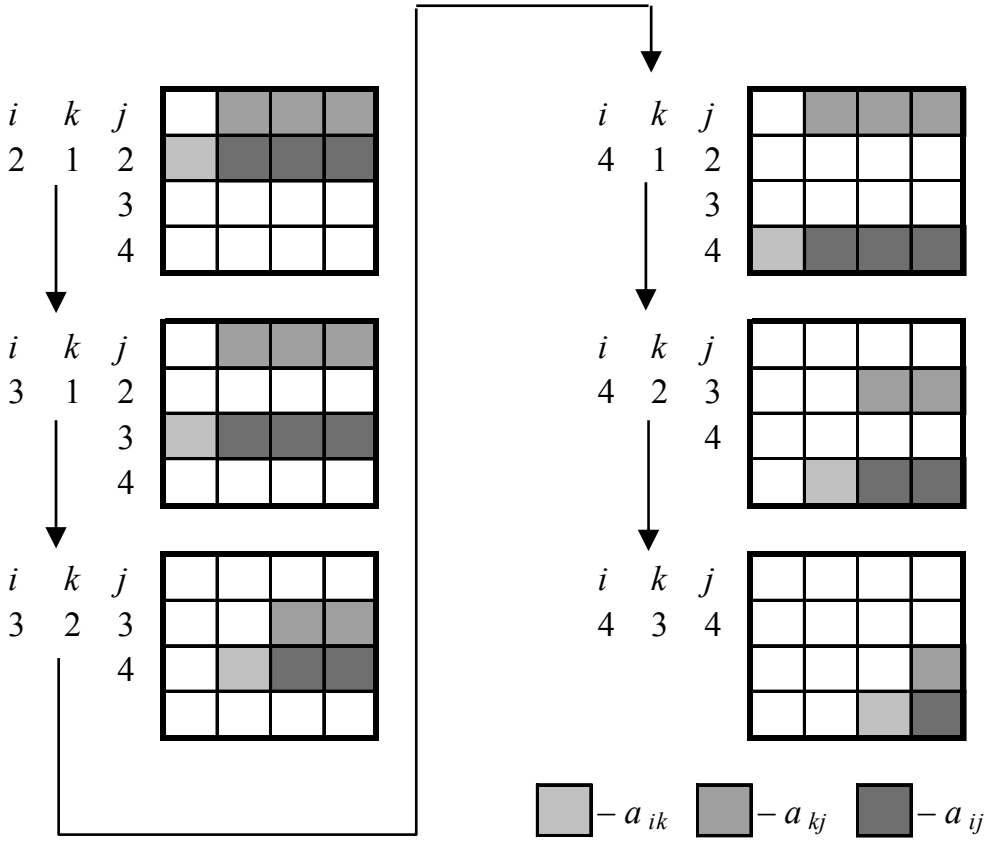


Рисунок 2.4 – Схематичное расположение элементов, используемых за все циклы ILU(0)-разложения

Алгоритм 2.5 – ILU(0)-разложение с использованием формата CSR без поиска ненулевых элементов

1 Для $i = 2, \ldots, N$
2 s_1 = **iptr**(i) – номер начального элемента
3 pr_1 = *Истина*
4 Пока pr_1 = *Истина* Продолжать
5 k = **jptr** (s_1)
6 Если $k \geq i$
7 Прервать текущий цикл
8 **aelem**(s_1) = **aelem**(s_1) / **aelem**(**diag**(k))
9 $s_2 = s_1$
10 $s_1 = s_1 + 1$
11 $y_1 = s_1$
12 y_{end1} = **iptr**(i+1)
13 y_2 = **diag**(k)+1

14 y_{end2} = **iptr**(k+1)
15 Если $y_{end1} \le y_1$ или $y_{end2} \le y_2$
16 Продолжить текущий цикл
17 pr_2 = *Истина*
18 Пока pr_2 = *Истина* Продолжать
19 Если **jptr**(y_1) = **jptr**(y_2)
20 **aelem**(y_1) = **aelem**(y_1) – **aelem**(s_2) × **aelem**(y_2)
21 $y_1 = y_1 + 1$
$y_2 = y_2 + 1$
22 Если **jptr**(y_1) > **jptr**(y_2)
23 $y_2 = y_2 + 1$
24 Если **jptr**(y_1) < **jptr**(y_2)
25 $y_1 = y_1 + 1$
26 Если **iptr**(k+1) < y_2 или **iptr**(i+1) < y_2
27 pr_2 = *Ложь*
28 Увеличить i

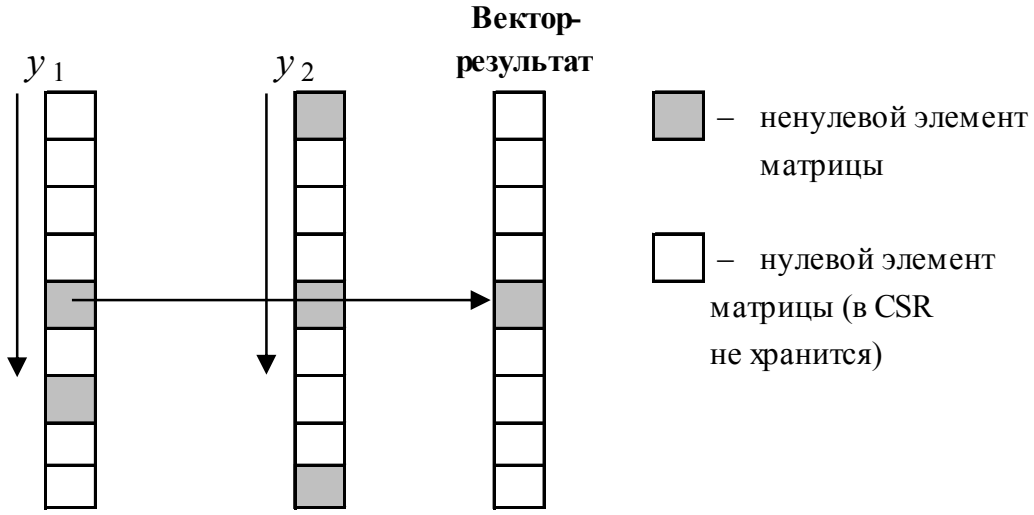


Рисунок 2.5 – Схематичное изображение операции сравнения ненулевых элементов в алгоритме 2.5

Можно воспользоваться временными векторами, так как в формате CSR хранятся не только сами элементы (массив **aelem**), но и адреса столбцов, в которых находятся эти элементы (массив **jptr**), что дает возможность построить циклы без дополнительных условий. Таким образом, суть предлагаемого усовершенствования заключается в том, что операция поиска одинаковых индексов столбцов двух векторов (строк) разделена на два этапа. На первом этапе создаются векторы **tmpvec** и **tmpjptr**: в первый записываются статусы наличия ненулевых элементов в анализируемом

векторе (строке), а во втором хранятся соответствующие адреса столбцов ненулевых элементов. На втором этапе сравниваются элементы второго вектора (строки) с элементами временного вектора (**tmpvec**), и если найдены ненулевые элементы с одинаковыми индексами столбцов, то выполняется соответствующая операция над элементами. Схематически это показано на рисунке 2.6.

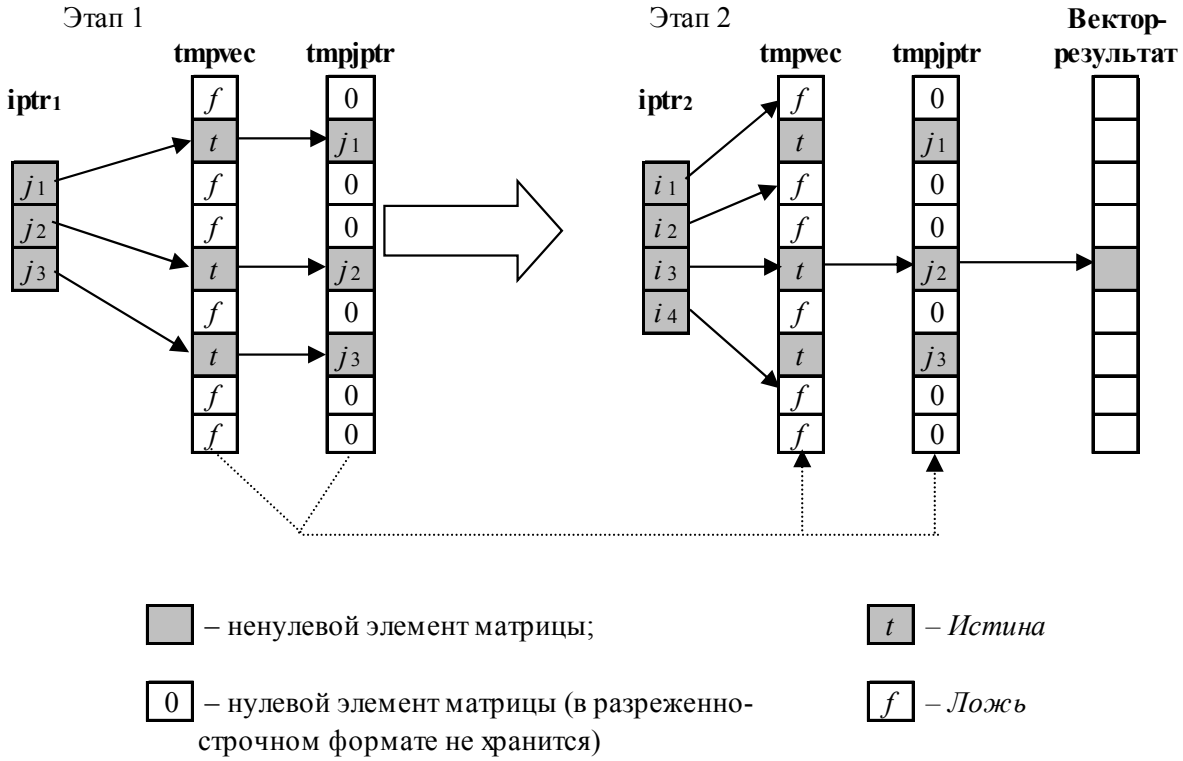


Рисунок 2.6 – Схематичное изображение этапов операции над ненулевыми элементами в алгоритме ILU(0)-разложения с использованием временных векторов

Очевидно, что выгоднее записывать во временные векторы данные строки *i*, поскольку она используется на протяжении всего шага по *i* (см. алгоритм 2.1), т.е. временные векторы будут создаваться один раз и использоваться на протяжении одного шага цикла *i*. С учетом этого разработан алгоритм 2.6. В нем в векторе **tmpvec** хранится значение *Истина*, если элемент ненулевой, а в векторе **tmpjptr** хранится адрес этого ненулевого элемента. Из приведенного алгоритма видно, что необходима дополнительная память компьютера для хранения *N* булевых элементов вектора **tmpvec** и *N* целочисленных значений их адресов в векторе **tmpjptr**. Такое увеличение машинной памяти по сравнению с па-

мятью для хранения основной матрицы незначительно и далее не учитывалось.

Алгоритм 2.6 – ILU(0)-разложение с использованием формата CSR с двумя дополнительными векторами

1 Для $i = 2, \ldots, N$
2 $s_1 =$ **iptr**(i) – номер начального элемента
3 $pr_1 =$ *Истина*
4 Для $j = s_1, \ldots,$ **iptr**(i+1)
5 **tmpvec**(**jptr** (j)) = *Истина*
6 **tmpjptr**(**jptr** (j)) = j
7 Увеличить j
8 Пока $pr_1 =$ *Истина* Продолжать
9 $k =$ **jptr** (s_1)
10 Если $k \geq i$
11 Прервать текущий цикл
12 **aelem**(s_1) = **aelem**(s_1) / **aelem**(**diag**(k))
13 $s_2 = s_1$
14 $s_1 = s_1 + 1$
15 $y_1 = s_1$
16 $y_{end1} =$ **iptr**(i+1)
17 $y_2 =$ **diag**(k)+1
18 $y_{end2} =$ **iptr**(k+1)
19 Если $y_{end1} \leq y_1$ или $y_{end2} \leq y_2$ Тогда
20 Продолжить текущий цикл
21 Для $j = y_2, \ldots, y_{end2}$
22 Если **tmpvec**(**jptr**(j)) = *Истина*
23 **aelem**(**tmpjptr**(**jptr**(j))) =
aelem(**tmpjptr**(**jptr**(j))) – **aelem**(s_2) × **aelem**(j)
24 Увеличить j
25 Увеличить i

2.2.2. Использование разреженного формата в итерационном методе

В итерационном методе при неявном предобусловливании на каждой итерации решается СЛАУ вида **My** = **z**. При этом матрица **M** хранится в формате CSR. Тогда за основу можно взять алгоритм 2.7 [24].

Алгоритм 2.7 – Решение СЛАУ с матрицей в формате CSR

1 Для $i = 1, \ldots, N$
2 $y_i = z_i$
3 Для $k =$ **iptrL**$(i), \ldots,$ **iptrL**$(i+1)$
4 $y_i = y_i -$ **aelemL**$(k)\, y_{\mathbf{jptrL}(k)}$
5 Увеличить k
6 $z_i = y_i /$ **aelemL**(**DiagL**(i))
7 Увеличить i
8 Для $i = N, \ldots, 1$
9 $z_i = y_i /$ **aelemU**(**DiagU**(i))
10 Для $k =$ **iptrU**$(i), \ldots,$ **iptrU**$(i+1)-1$
11 $y_i = y_i -$ **aelemU**$(k)\, y_{\mathbf{jptrU}(k)}$
12 Увеличить k
13 Уменьшить i

Строки 1–7 — прямой ход, строки 8–13 — обратный ход. Циклы в строках 3 и 10 выполняют обход по элементам матрицы. Приведенный алгоритм требует разделения на подматрицы **L** (массивы **aelemL**, **iptrL** и **jptrL**) и **U** (массивы **aelemL**, **iptrL** и **jptrL**). Однако в рассматриваемом случае такого разделения нет. С учетом этой особенности был разработан алгоритм 2.8.

Алгоритм 2.8 – Решение СЛАУ с матрицей в модифицированном формате CSR

1 Для $i = 1, \ldots, N$
2 $s_1 =$ **iptr**(i)
3 $s_2 =$ **iptr**$(i+1)$
4 $y_i = z_i$
5 Для $k = s_1, \ldots, s_2$
6 Если **jptr**$(k)<i$
7 $y_i = y_i -$ **aelem**$(k)\, y_{\mathbf{jptr}(k)}$
8 Увеличить k
9 Увеличить i
10 Для $i = N, \ldots, 1$
11 $s_1 =$ **iptr**(i)
12 $s_2 =$ **iptr**$(i+1)$
13 Для $k = s_1, \ldots, s_2$
14 Если **jptr**$(k)>i$
15 $y_i = y_i -$ **aelem**$(k)\, y_{\mathbf{jptr}(k)}$
16 Увеличить k
17 $y_i = y_i /$ **aelem**(**Diag**(i))
18 Уменьшить i

Строки 1–9 — прямой ход, 10–18 — обратный ход. Циклы в строках 5 и 13 выполняют обход по элементам матрицы. Условие в строке 6 обеспечивает обход только элементов нижней треугольной подматрицы **L**, а условие в строке 14 — верхней треугольной подматрицы **U**. Тем самым обеспечивается решение без разделения на подматрицы **L** и **U**, как это сделано в алгоритме 2.7.

2.3. Вычислительные эксперименты

Для подтверждения эффективности предложенных алгоритмов проведены вычислительные эксперименты [127, 128]. Использовался персональный компьютер (при вычислениях не применялось распараллеливание, т.е. работало одно ядро процессора) с параметрами: платформа AMD Athlon(tm) 64 X2 Dual; частота процессора 2100 МГц; объем ОЗУ 2 Гбайта; число ядер — 2; операционная система Windows XP.

Для подтверждения эффективности использования дополнительного вектора **Diag** проведено сравнение времени работы алгоритма 2.2 (T) и алгоритма 2.4 (T_D) при различной плотности q матрицы **M** порядка $N = 1000$. Разная плотность матрицы **M** получена с помощью предфильтрации (1.24) с различным порогом обнуления ε. Результаты сравнения представлены в таблице 2.4. Как видно, использование вектора **Diag** ускоряет ILU(0)-разложение в 1,14–1,23 раза.

Далее сравнивались алгоритмы 2.1 (ILU(0)-разложение, *ikj*-версия) и 2.4 (ILU(0)-разложение с использованием формата CSR с дополнительным вектором **Diag**). Для реализации выбран итерационный метод BiCGStab, так как используемый совместно с неявным предобусловливанием, он хорошо зарекомендовал себя при решении СЛАУ с плотной матрицей [58, 129]. Измерялось время трех составляющих полного решения СЛАУ: предфильтрации (в алгоритме 2.4, использующем разреженный формат, в ходе предфильтрации выполняется и конвертация матрицы в формат CSR [127]); ILU(0)-разложения; итерационного процесса.

Оценка временных затрат сделана для трех СЛАУ, матрицы которых получены в системе TALGAT последовательным учащением сегментации границ проводник-диэлектрик и диэлектрик-

диэлектрик из задачи вычисления электрической емкости двух полосок на диэлектрическом слое над идеально проводящей плоскостью (рисунок 2.7), для двух вариантов алгоритма BiCGStab: без использования формата хранения разреженных матриц (*T*) и с использованием формата CSR с дополнительным вектором **Diag** (*Tcsrd*).

Таблица 2.4 – Сравнение времени выполнения алгоритмов 2.2 и 2.4

ε	*q*, %	*T*, с	T_D, с	T / T_D
0,012	59	285	258	1,14
0,014	56	261	234	1,13
0,016	53	242	213	1,16
0,018	50	224	195	1,15
0,020	47	207	178	1,16
0,022	45	192	162	1,17
0,024	42	175	146	1,17
0,026	40	163	133	1,17
0,028	38	152	123	1,18
0,030	36	142	112	1,19
0,032	35	132	103	1,19
0,034	33	124	95	1,20
0,036	31	116	88	1,20
0,038	30	109	81	1,20
0,040	29	103	75	1,21
0,042	27	97	70	1,21
0,044	26	91	65	1,23
0,046	25	86	60	1,22
0,048	24	81	56	1,23
0,050	23	73	52	1,23

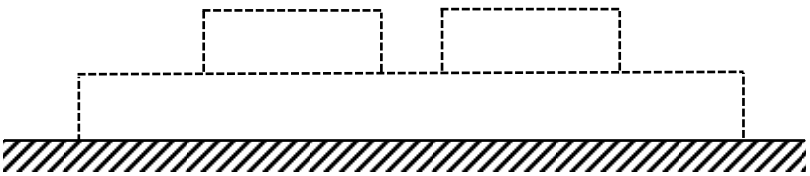


Рисунок 2.7 – Исследуемая конфигурация – два проводника с диэлектриком над идеально проводящей плоскостью

В экспериментах использовались матрицы порядка $N = 4800$, 6000, 8000. При помощи предфильтрации (1.24) изменялся порог обнуления ε, меняющий плотность матрицы **M**. Так, для $N = 4800$ порог менялся в диапазоне 0, $1{,}5 \cdot 10^{-6}$, …, $1{,}75 \cdot 10^{-5}$; для $N = 6000$ — 0, $1{,}0 \cdot 10^{-7}$, …, $1{,}0 \cdot 10^{-5}$; для $N = 8000$ — 0, $5{,}0 \cdot 10^{-3}$, …, $1{,}0 \cdot 10^{-1}$. Итерации продолжались, пока относительная норма вектора невязки не становилась меньше 10^{-6}.

Сравнение времени двух алгоритмов для различных порядков матриц приведено на рисунке 2.8. Видно, что ускорение алгоритма с использованием формата CSR получается не при всех значениях плотности, а только для значений, меньших определенного порогового значения. При этом максимальное ускорение от использования формата хранения разреженной матрицы достигается приблизительно при оптимальном значении порога обнуления, соответствующем минимальному времени решения СЛАУ. В таблице 2.5 приведены данные сравнения алгоритмов при различных порядках матриц. Видно, что ускорение имеет место на различных матрицах, изменяясь в пределах от 1,5 до 1,6 раза. Дополнительно приведено время решения методом Гаусса (*Tge*) и ускорение относительно него, изменяющееся в пределах от 1,2 до 4,4 раза (использовался алгоритм решения методом Гаусса, основанный на LU-разложении и последующем решении.

Если проанализировать время работы алгоритма решения СЛАУ по составляющим, то выясняется, что ускорение получается именно за счет вычисления ILU(0)-разложения. На рисунке 2.8 приведено время работы алгоритма ILU(0)-разложения для $N = 4800$ (рисунок 2.8,*б*), $N = 6000$ (рисунок 2.8,*г*), $N = 8000$ (рисунок 2.8,*е*).

Вычисление ILU(0)-разложения занимает часть общего времени решения СЛАУ, следовательно, его сокращение обеспечивает общее ускорение. Ускорение ILU(0)-разложения достигается за счет использования формата CSR, так как в данном формате нулевые элементы матрицы не хранятся, а следовательно, не участвуют в алгоритме, в отличие от алгоритма, который работает с полной матрицей.

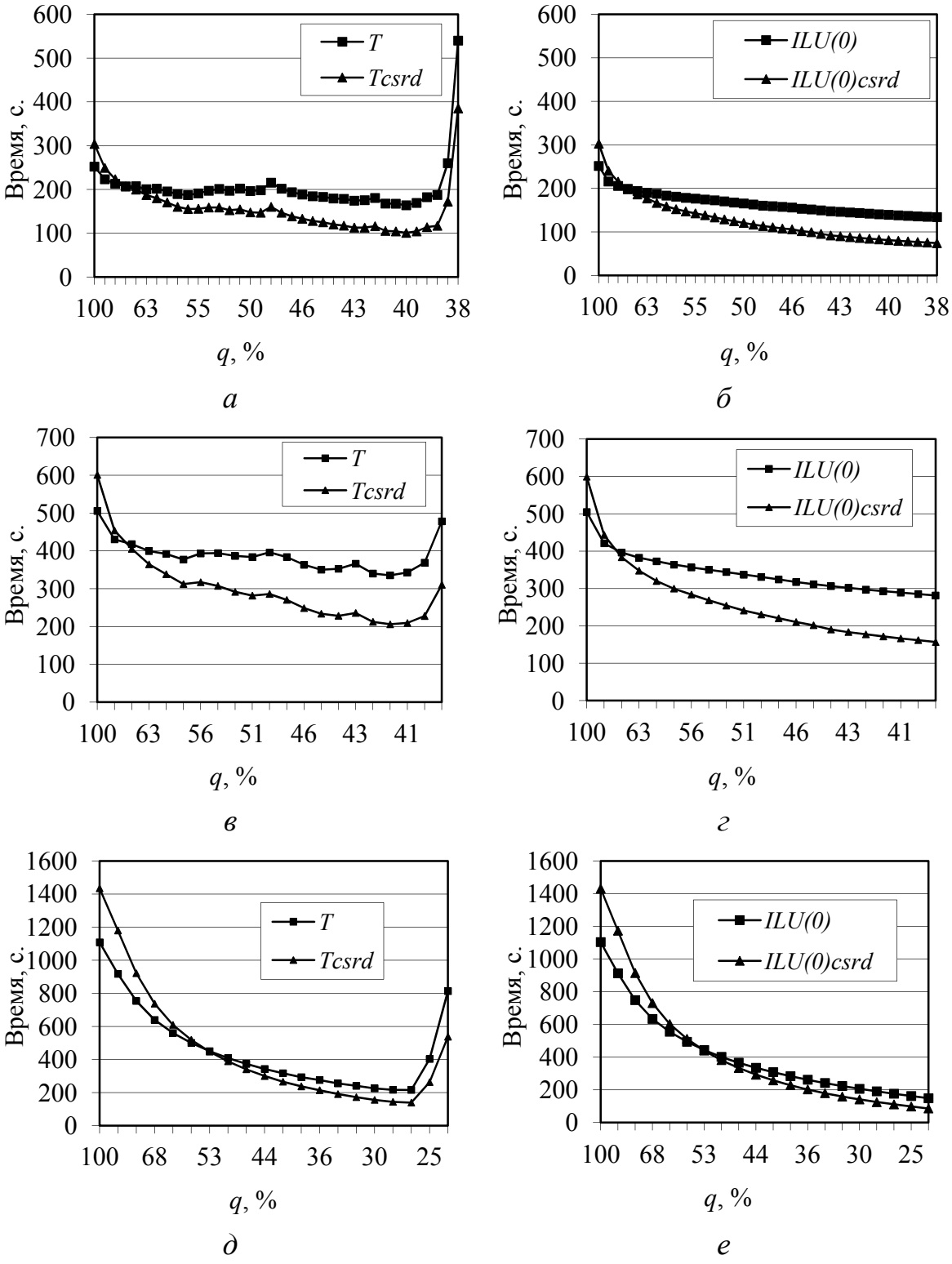


Рисунок 2.8 – Сравнение времени работы алгоритмов:
без использования формата хранения разреженных матриц
(*T* – время решения СЛАУ методом BiCGStab, ILU(0) – время вычисления ILU(0)-разложения); с использованием формата CSR с дополнительным вектором **Diag** (Tcsrd – время решения СЛАУ методом BiCGStab, ILUcsrd (0) – время вычисления ILU(0)-разложения)

Для вычислительного эксперимента с последующими алгоритмами использовался персональный компьютер (при вычислениях не применялось распараллеливание, т.е. работало одно ядро процессора) с параметрами: платформа Intel(R) Core(TM) i7 CPU 970; частота процессора 3,20 ГГц; объем ОЗУ 16 Гбайт; число ядер — 6; операционная система Linux Kubuntu11.10 x64.

Таблица 2.5 – Сравнение времени решения СЛАУ разными алгоритмами

N	*T*, с	*Tcsrd*, с	*T*/*Tcsrd*	*Tge*, с	*Tge*/*Tcsrd*
4800	164	101	1,6	129	1,3
6000	335	206	1,6	254	1,2
8000	216	140	1,5	616	4,4

Оценка временных затрат сделана на нескольких СЛАУ (полученных последовательным учащением сегментации границ проводник-диэлектрик и диэлектрик-диэлектрик из задачи вычисления электрической емкости), решаемых методом BiCGStab с использованием предобусловливания по алгоритмам 2.4, 2.5 и 2.6. Итерации продолжались, пока относительная норма вектора невязки была больше 10^{-6}. Для оценки ускорения решения СЛАУ использовано отношение времени решения СЛАУ по усовершенствованным алгоритмам 2.5 (*Ts*), 2.6 (*Tr*) ко времени решения алгоритма 2.4 (*T*). Вид конфигурации 1, состоящей из двух проводников на слое диэлектрика над идеально проводящей плоскостью, приведен на рисунке 2.7, а конфигурации 2, состоящей из двух проводников (без диэлектрика) над идеально проводящей плоскостью, — на рисунке 2.9. Полученные зависимости ускорения решения СЛАУ от плотности *q* матрицы **M** приведены на рисунке 2.10.

Рисунок 2.9 – Исследуемая конфигурация – два проводника над идеально проводящей плоскостью

Из рисунка 2.10 следует, что усовершенствованные алгоритмы 2.5 и 2.6 работают быстрее алгоритма 2.4. Максимальные ускорения получены при $N = 4800$: в 1,6 и 2,5 раза. При $N = 8000$ максимальные ускорения составили 1,2 и 1,7 раза соответственно.

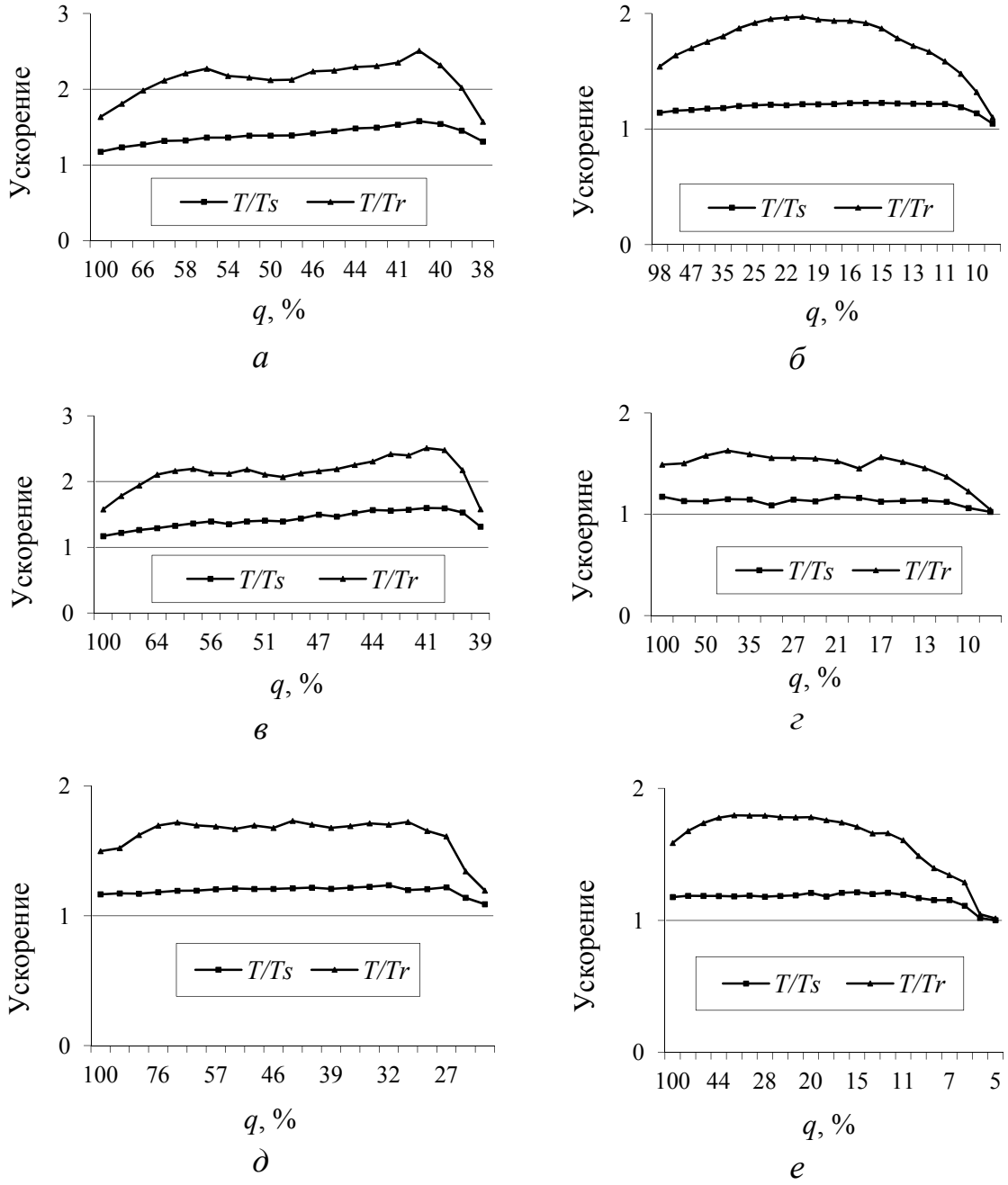


Рисунок 2.10 – Ускорение решения СЛАУ методом BiCGStab при использовании алгоритма 2.4 (*T*) относительно алгоритма 2.5 (*Ts*) и алгоритма 2.6 (*Tr*) для конфигурации 1 (рисунок 2.7) при $N = 4800$ (*а*), 6000 (*в*), 8000 (*д*) и конфигурации 2 (рисунок 2.9) при $N = 2240$ (*б*), 6800 (*г*), 7999 (*е*)

Также видно, что при малых плотностях матрицы **M** ускорение падает. Этот факт объясняется тем, что в данной работе оптимизировался только этап ILU(0)-разложения, т.е. одна из трех составляющих общего времени: предфильтрации (*Tp*), ILU(0)-разложения (*Tlu*) и итерационного процесса (*Ti*). На рисунке 2.11 приведено ускорение ILU(0)-разложения при использовании алгоритмов 2.5 (*Tlus*) и 2.6 (*Tlur*) относительно алгоритма 2.4 (*Tlu*).

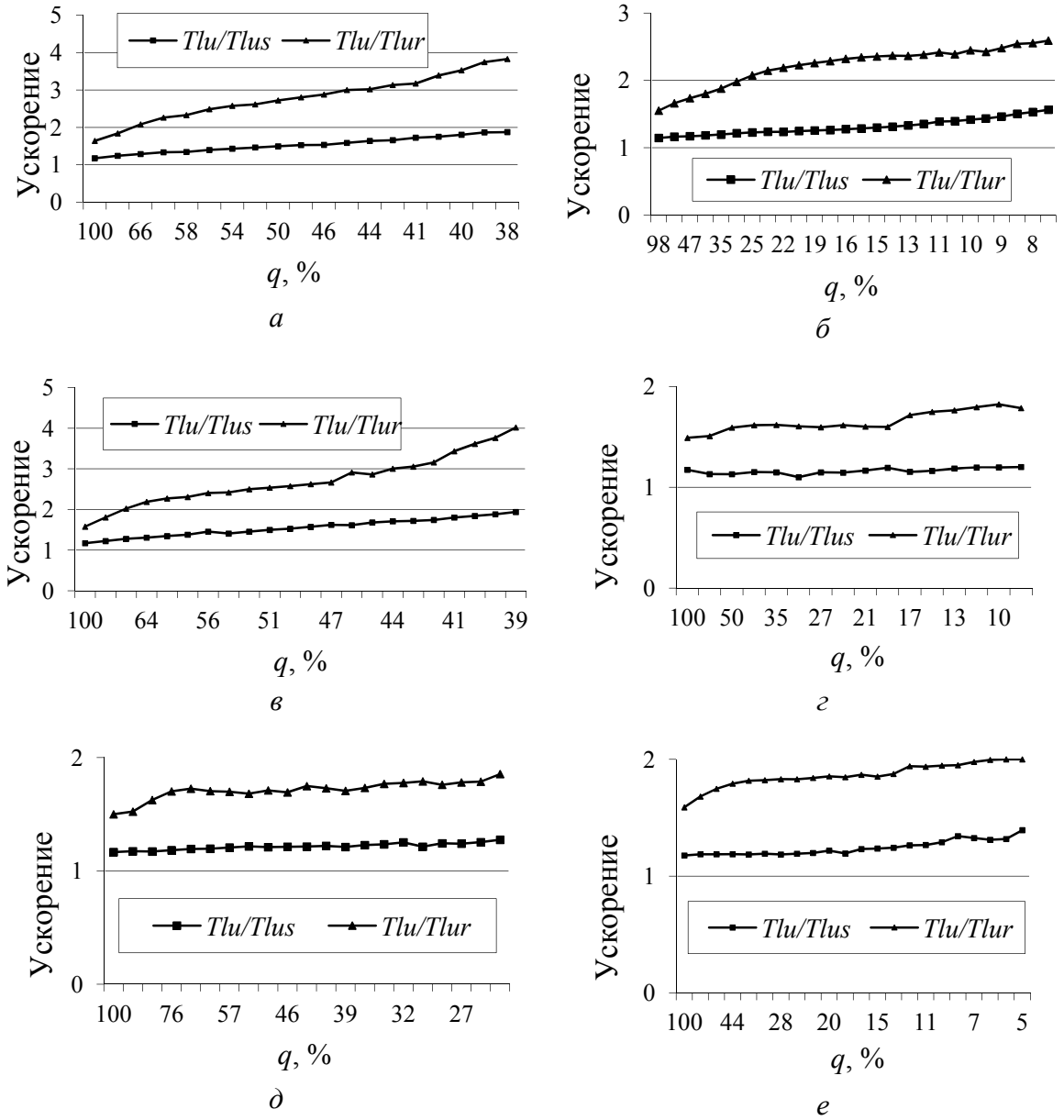


Рисунок 2.11 – Ускорение ILU(0)-разложения при использовании алгоритма 2.4 (*Tlu*) относительно алгоритма 2.5 (*Tlus*) и алгоритма 2.6 (*Tlur*) для конфигурации 1 (рисунок 2.7) при $N = 4800$ (*а*), 6000 (*в*), 8000 (*д*) и для конфигурации 2 (рисунок 2.9) при $N = 2240$ (*б*), 6800 (*г*), 7999 (*е*)

Видно, что для $N = 4800$ максимальное ускорение для алгоритма 2.5 составляет 2 раза, а для алгоритма 2.6 — почти 4 раза. На всех графиках ускорение увеличивается с уменьшением плотности матрицы. Нет отсутствия ускорения при минимальной плотности матрицы, как это наблюдается на рисунке 2.10.

Показательно сравнение ускорения на разных этапах решения при использовании алгоритмов 2.4 и 2.6. Для наглядности зависимость соотношения времени этапов решения СЛАУ по алгоритму 2.6 от плотности q матрицы **M** для $N = 4800$ изображена на рисунке 2.12.

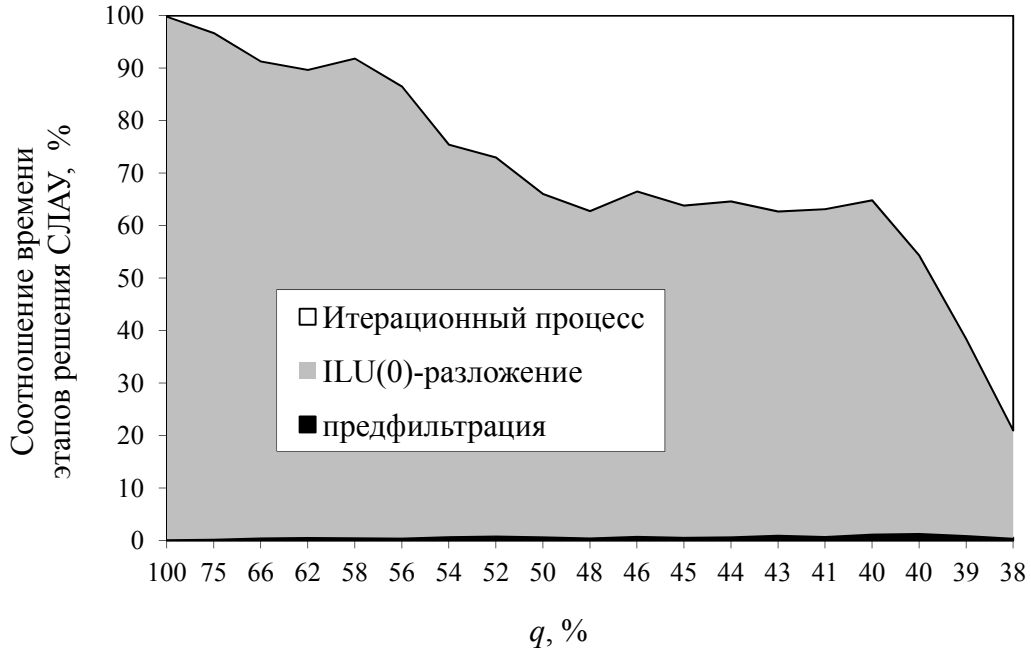


Рисунок 2.12 – Зависимость соотношения времени, затрачиваемого на этапы решения СЛАУ итерационным методом с предобусловливанием, от плотности матрицы **M** для $N = 4800$

Из рисунка 2.12 видно, что при уменьшении плотности матрицы часть времени, затрачиваемая на итерационный процесс, увеличивается, а на ILU(0)-разложение (за счет которого получено ускорение) — уменьшается. Следовательно, ускорение общего времени решения СЛАУ также уменьшается. Между тем снижение ускорения при малой плотности матрицы не уменьшает значимость усовершенствования алгоритма. Действительно, на практике требуется минимизировать время решения СЛАУ за счет

выбора оптимального значения порога обновления. При этом время ILU(0)-разложения примерно равно времени итерационного процесса [130]. Следовательно, уменьшение времени вычисления ILU(0)-разложения весьма важно.

3. Многократное решение СЛАУ с изменяющейся матрицей итерационными методами с предобусловливанием

3.1. Многократное решение СЛАУ итерационными методами без переформирования предобусловливателя

3.1.1. Подходы к использованию итерационных методов при многократном решении СЛАУ

Для многократного решения СЛАУ с изменяющейся матрицей [131, 132] использован итерационный метод с предобусловливателем, полученным из матрицы первой СЛАУ.

Алгоритм 3.1 – Многократное решение СЛАУ итерационным методом с предобусловливанием

1 Получить матрицу $\mathbf{A}_S$ из матрицы $\mathbf{A}_1$ с помощью предфильтрации
2 Вычислить матрицу $\mathbf{M}$ из матрицы $\mathbf{A}_S$
3 Для k от 1 до m
4 Итерационно вычислить $\mathbf{x}_i$ из уравнения $\mathbf{MA}_k\mathbf{x}_k = \mathbf{Mb}$ с заданной точностью *Tol*
5 Увеличить k

В данном алгоритме в качестве способа формирования матрицы предобусловливания использовано ILU(0)-разложение. Оценим возможное ускорение от применения итерационного метода. Если представить его отношением общего времени решения m СЛАУ прямым методом (T_D) ко времени вычисления по алгоритму 3.1, то получим

$$\beta = \frac{mT_D}{T_{PR} + \sum_{k=1}^{m} T_k}, \qquad (3.1)$$

где T_{PR} — время формирования матрицы предобусловливания; T_k — время итерационного вычисления $\mathbf{x}_k$ с заданной точностью.

Предположим, что при решении отдельных СЛАУ T_k существенно не изменяется и в среднем равно T_A. Тогда из (3.1) получим усредненное ускорение

$$\beta_A = \frac{mT_D}{T_{PR} + mT_A} \tag{3.2}$$

и оценку максимального усредненного ускорения

$$\beta_A^{\max} = \lim_{m\to\infty}\left(\frac{mT_D}{T_{PR} + mT_A}\right) = \frac{T_D}{T_A}. \tag{3.3}$$

Формулы (3.1) и (3.3) имеют важные следствия.

Следствие 1. Чем больше число решаемых СЛАУ, тем меньше ускорение зависит от времени построения предобусловливателя, а значит, от выбора вида предобусловливания, способа предфильтрации и оптимального значения допуска обнуления.

Следствие 2. Поскольку максимальное усредненное ускорение обратно пропорционально среднему времени итерационного процесса, то актуально уменьшение времени одной итерации и числа итераций.

При использовании предобусловливания время итерационного процесса зависит от значения допуска обнуления τ: чем оно меньше, тем, вообще говоря, итерационный процесс быстрее. Это происходит за счет формирования более плотной матрицы **M**, что, как правило, ведет к уменьшению числа итераций [50]. Оно, очевидно, минимально при $\tau = 0$. Это следствие делает неэффективным использование форматов хранения разреженных матриц, так как отсутствует экономия памяти.

Еще одним параметром, влияющим на время итерационного процесса, является начальное приближение $\mathbf{x}^0$. Ясно, что чем $\mathbf{x}^0$ ближе к решению системы, тем меньше итераций потребуется итерационному методу для его уточнения. Исходя из этого следует предположить, что для вектора начального приближения мож-

но использовать вектор решения предыдущей СЛАУ. Для подтверждения этого сравнивались два варианта выбора $\mathbf{x}^0$: фиксированное начальное приближение (все элементы вектора равны некоторому значению, в данном случае единице), используемое для решения каждой СЛАУ; вектор решения предыдущей СЛАУ.

Полезно оценить изменение элементов текущей матрицы СЛАУ относительно первой. Так как в нашем случае изменяющиеся элементы находятся на главной диагонали и равны между собой, рассматривались значения одного элемента матрицы. Зависимость изменения значений Δa от k приведена на рисунке 3.1. Матрицы получены для структуры, изображенной на рисунке 2.7. Видно, что изменение значений элементов СЛАУ с увеличением k становится меньше и не превышает 12 %. Таким образом, для данной задачи представляется более выгодным использовать вектор решения предыдущей СЛАУ.

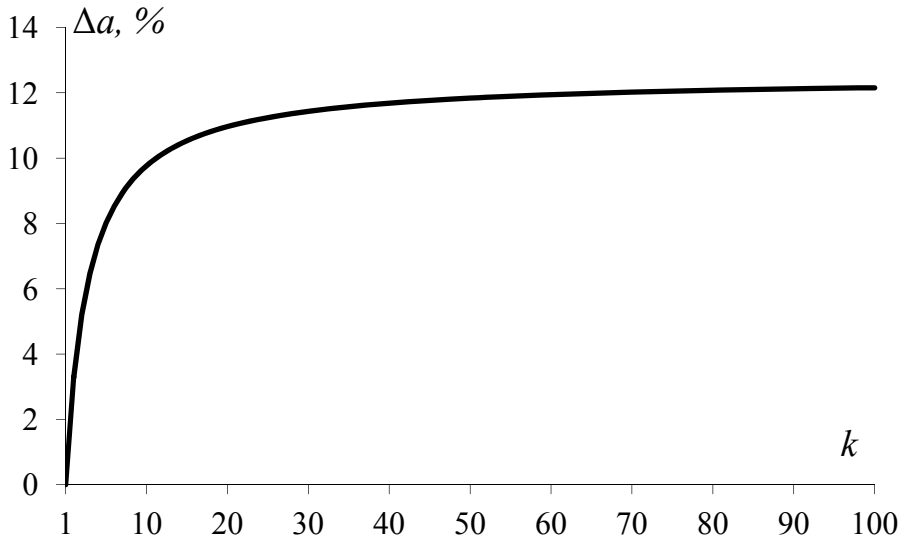


Рисунок 3.1 – Зависимость Δa от k

Часто необходимо моделирование при изменении размеров структуры, что приводит к изменению элементов матрицы, расположенных в произвольных местах [126]. В данном случае алгоритм, использующий блочное LU-разложение, не применим. Поэтому была рассмотрена возможность многократного решения СЛАУ итерационным методом (см. алгоритм 3.1).

Для ускорения итерационного процесса рассматривались два способа. Первый — это использование в качестве вектора начального приближения $\mathbf{x}^0$ вектора решения предыдущей СЛАУ, т.е. $\mathbf{x}_k^0 = \mathbf{x}_{k-1},\ k = 1, 2,\ \ldots,\ m$ (для первой СЛАУ использовался единичный вектор). Второй способ — это использование матрицы предобусловливания **M**, полученной при решении первой СЛАУ, т.е. $\mathbf{M}_k = \mathbf{M}_1$.

3.1.2. Вычислительный эксперимент

Для вычислительного эксперимента использовался персональный компьютер (при вычислениях распараллеливание не применялось, т.е. работало одно ядро процессора) со следующими параметрами: платформа AMD FX(tm)-8320 Eight-Core Processor; частота процессора 3,50 ГГц; объем ОЗУ 16 Гбайт; число ядер — 8; операционная система Windows 7x64.

В качестве исследуемой использована простая структура (см. рисунок 2.7), полученная в системе TALGAT. Целью моделирования являлась оценка временных затрат, необходимых для вычисления m емкостных матриц. В эксперименте использовалась одна структура, но посредством изменения сегментации границ проводников и диэлектрика были получены СЛАУ различных порядков (N = 708, 1416, 3540, 4425). В качестве итерационного метода применялся метод BiCGStab [127, 133]. Предобусловливатель **M** получен с помощью LU-разложения исходной матрицы. Используемые матрицы были проанализированы, результаты анализа представлены в таблице 3.1 (число обусловленности рассчитано по формуле (1.16)). На рисунке 3.2 приведены портреты матриц, которые получены следующим образом: все элементы матрицы сортируются по модулю значений, затем разбиваются на четыре группы, каждой группе соответствует свой цвет. Как видно, матрицы имеют схожий характер, поскольку были получены увеличением сегментации.

При вычислениях использовался алгоритм 3.1 с двумя вариантами начального приближения: фиксированным и равным вектору решения предыдущей СЛАУ. Итерационный процесс про-

должался, пока относительная норма вектора невязки была больше 10^{-6}.

Выбор критерия остановки 10^{-6} для данной задачи специально не исследовался. Однако примеры решения одиночных СЛАУ, возникающих в электродинамических задачах, как правило, гораздо хуже обусловленных, показывают, что относительная точность 10^{-6} по норме невязок обеспечивает 7–8 точных знаков после запятой в решении [130].

Таблица 3.1 – Характеристики матриц для структуры, представленной на рисунке 2.7

N	N_c	Плотность матрицы **A**, %	Плотность матрицы **M**, %	Число обусловленности
708	288	100	100	$1{,}75 \cdot 10^8$
1416	576	100	100	$5{,}78 \cdot 10^8$
3540	1440	100	100	$2{,}19 \cdot 10^9$
4425	1800	100	100	$3{,}12 \cdot 10^9$

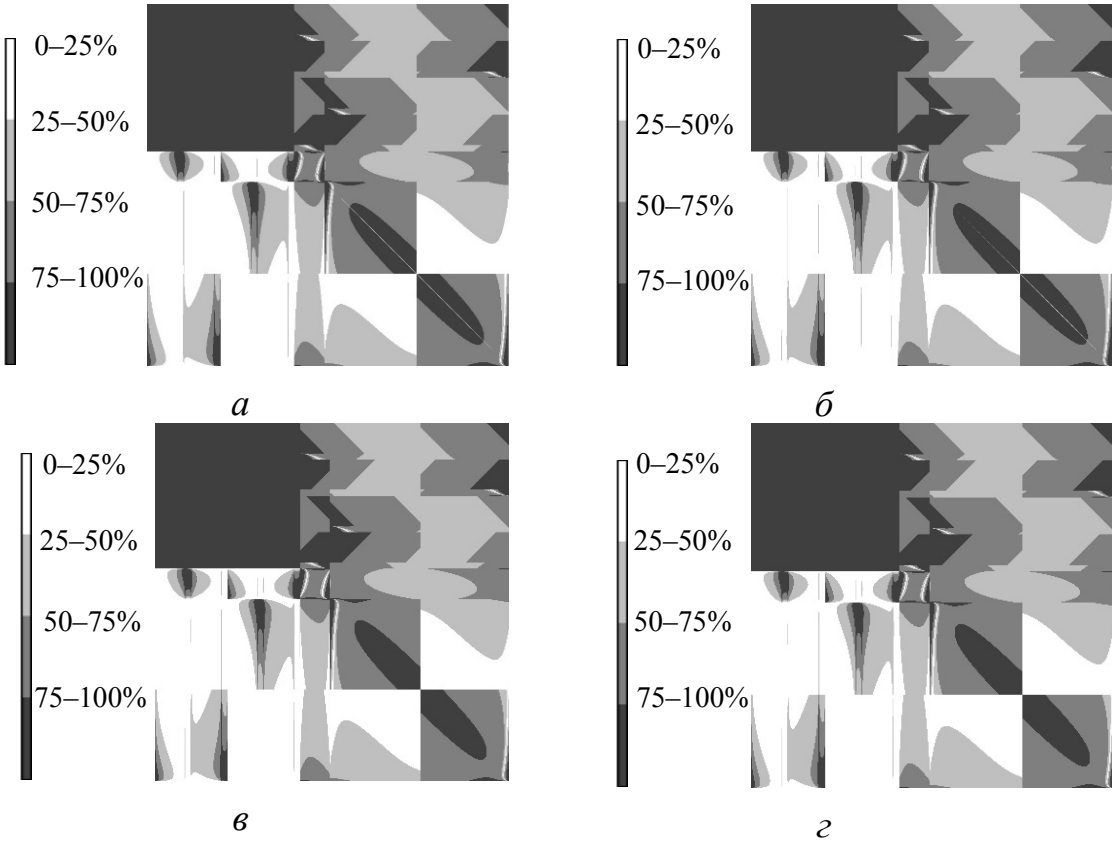


Рисунок 3.2 – Портреты матриц при разных N: 708 (*а*); 1416 (*б*); 3540 (*в*); 4425 (*г*)

Как показали многочисленные эксперименты (на каждой из исследуемых матриц с учетом их изменения и обоих вариантов начального приближения), максимальное абсолютное значение элементов разности векторов решения, полученных методом Гаусса и итерационным методом, не превышает 10^{-4}. Между тем для практики, как правило, приемлема точность 1–2 знака [5]. Число итераций при стократном решении приведено на рисунке 3.3.

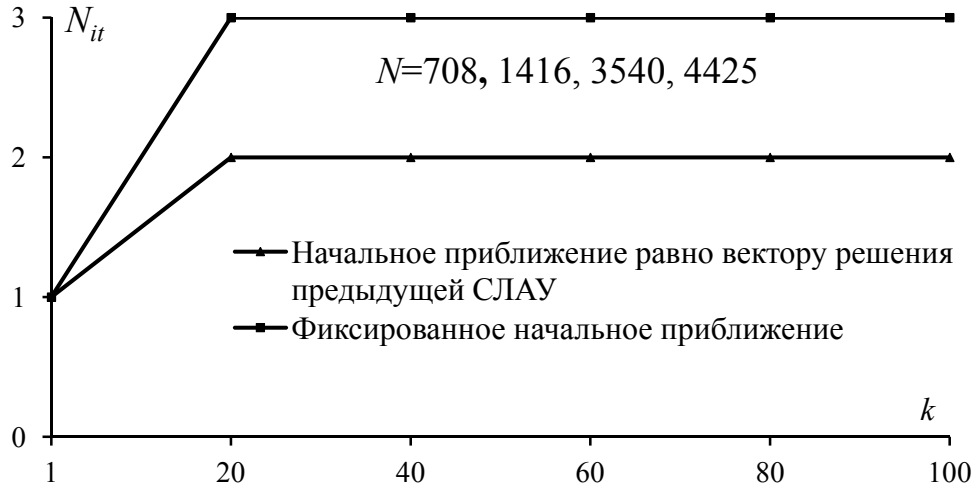


Рисунок 3.3 – Зависимости числа требуемых итераций для решения одной СЛАУ от k при разных вариантах начального приближения (совпадают для всех N)

Как видно, использование решения предыдущей СЛАУ уменьшает число итераций. Далее проведены вычисления с целью оценки ускорения решения итерационным методом относительно метода Гаусса. Результаты для решения m СЛАУ (с разными N) с помощью алгоритма 3.1 приведены в таблице 3.2. Они показывают ускорение при $m > 1$. С ростом m ускорение растет, достигая 49 при $m = 1000$ и $N = 4425$. В последней строке таблицы приведены значения, полученные по формуле (3.3). Видно, что к ним сходятся значения предыдущих строк, что подтверждает возможность аналитических оценок ускорений по формуле (3.3) без вычислительного эксперимента. Также видно, что начальное приближение, равное вектору решения предыдущей СЛАУ, дает большее ускорение, чем фиксированное (в 1,4 раза при $N = 4425$).

Далее исследовалась структура из одного проводника на диэлектрической подложке над идеально проводящей плоскостью (рисунок 3.4).

Таблица 3.2 – Ускорение (относительно метода Гаусса) решения итерационным методом m СЛАУ

m	Фиксированное начальное приближение				Начальное приближение равно вектору решения предыдущей СЛАУ			
	$N = 708$	$N = 1416$	$N = 3540$	$N = 4425$	$N = 708$	$N = 1416$	$N = 3540$	$N = 4425$
1	0,36	0,51	0,52	0,31	0,36	0,51	0,52	0,31
5	1,37	2,15	2,43	1,52	1,53	2,27	2,49	1,53
10	2,12	3,65	4,53	2,93	2,52	4,00	4,74	2,98
100	4,12	9,58	20,53	18,09	5,70	12,74	24,82	20,42
1000	4,55	11,43	31,71	37,36	6,59	16,32	43,09	49,3
∞	4,61	11,69	33,78	42,43	6,71	16,86	46,96	58,59

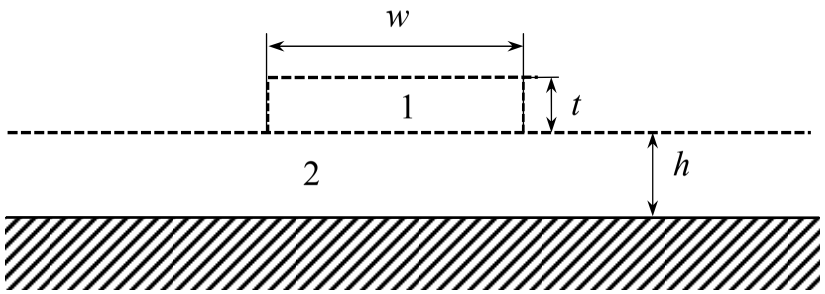


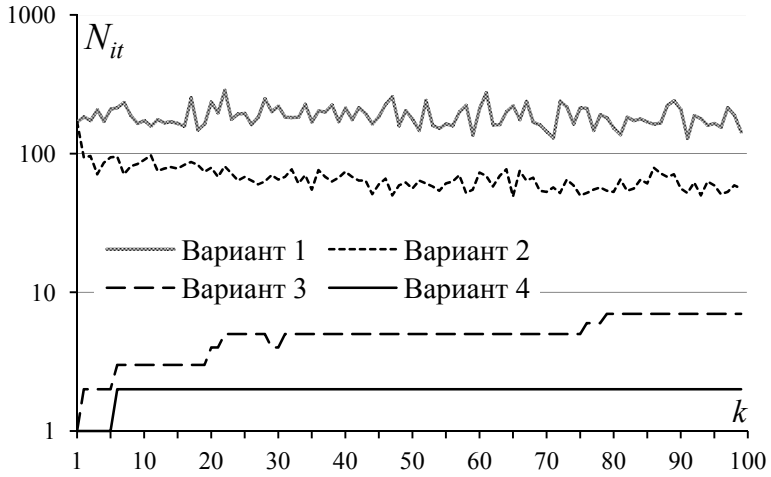
Рисунок 3.4 – Вид поперечного сечения исследуемой структуры в системе TALGAT (проводник 1 на диэлектрической подложке 2 над идеально проводящей плоскостью)

Целью эксперимента являлась оценка временных затрат для вычисления 100 емкостных матриц, полученных путем изменения одного из размеров структуры: высоты диэлектрика h (в диапазоне 12–112 мкм, или на 933 %); ширины проводника w (в диапазоне 18–118 мкм, или на 656 %); высоты проводника t (в диапазоне 6–106 мкм, или на 1767 %). Количество сегментов на каждом отрезке структуры не менялось, что обеспечивало постоянство порядка (1600) матриц СЛАУ, необходимое для корректного сравнения. Итерации продолжались, пока относительная норма вектора невязки была больше 10^{-8}. Для сравнения использовался метод исключения Гаусса.

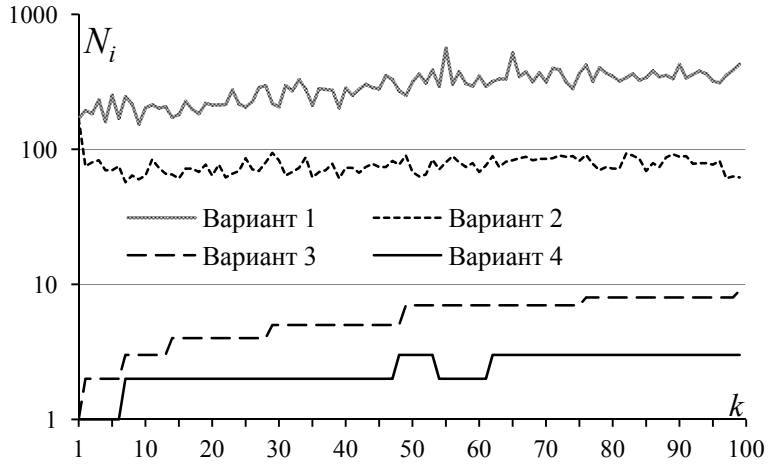
Для подтверждения эффективности способов ускорения предложено сравнить следующие варианты вычислений. В исходном варианте 1 ускорение не использовано. В вариантах 2, 3 способы ускорений использованы отдельно, а в варианте 4 — совместно. На рисунке 3.5 приведено изменение числа итераций N_{it} при решении k-й СЛАУ для каждого из вариантов.

Анализ полученных данных позволяет сделать несколько выводов. В варианте 1 количество итераций велико и в среднем постоянно, поскольку не используются способы ускорения решения. В варианте 2, где в качестве начального приближения выбран вектор решения предыдущей СЛАУ, наблюдается небольшое и постепенное уменьшение количества итераций, а следовательно, времени решения СЛАУ.

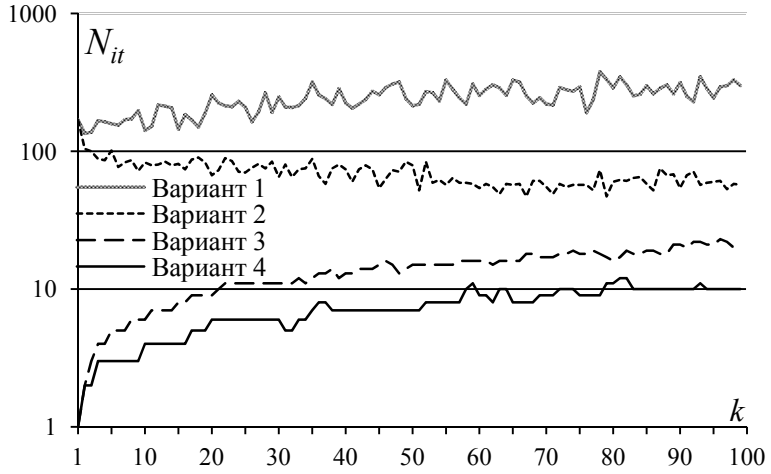
В варианте 3 используется матрица предобусловливания, сформированная из матрицы первой СЛАУ. При решении первой СЛАУ количество итераций равно 1, а при решении последующих СЛАУ оно растет из-за изменений в матрице. В варианте 4 количество итераций минимально, что приводит к снижению общего времени решения всех СЛАУ и доказывает эффективность совместного использования способов ускорения. Также следует отметить, что при изменении разных размеров число итераций для решения СЛАУ разное. При изменении h и w количество итераций меньше, чем при изменении t. Это может быть связано с гораздо большими изменениями в матрице СЛАУ.



а



б



в

Рисунок 3.5 – Число итераций при решении k-й СЛАУ методом BiCGStab для вариантов 1–4 при изменении h (*а*); w (*б*); t (*в*)

Для оценки изменений в матрицах СЛАУ использовались нормы матриц изменений $\|\Delta\mathbf{A}_k\|_1$ и $\|\Delta\mathbf{A}_k\|_\infty$, где $\Delta\mathbf{A}_k$ — матрица изменений ($\Delta\mathbf{A}_k = \mathbf{A}_k - \mathbf{A}_1$).

Расчет норм выполнялся по формулам

$$\|\Delta\mathbf{A}_k\|_1 = \max_i \sum_j |a_{ij}|, \quad \|\Delta\mathbf{A}_k\|_\infty = \max_j \sum_i |a_{ij}|.$$

Результаты вычислений отношения норм матриц $\|\Delta\mathbf{A}_k\|_1 / \|\Delta\mathbf{A}_2\|_1$, $\|\Delta\mathbf{A}_k\|_\infty / \|\Delta\mathbf{A}_2\|_\infty$ приведены на рисунке 3.6. Видно, что для *t* изменения больше, чем для *h* и *w*.

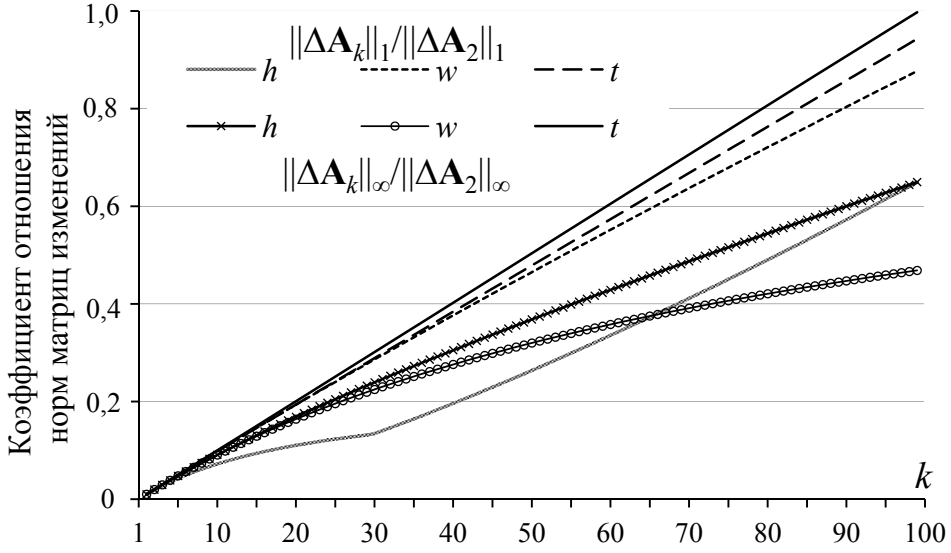


Рисунок 3.6 – Нормированные (по $\|\Delta\mathbf{A}_{100}\|_\infty / \|\Delta\mathbf{A}_2\|_\infty$ при изменении *t*) зависимости отношений норм матриц от *k*, полученных при изменении размеров *h*, *w*, *t*

Полученная матрица была проанализирована, результаты анализа представлены в таблице 3.3. На рисунке 3.7 приведен портрет матрицы при *h* = 12 мкм, *w* = 18 мкм, *t* = 6 мкм.

Таблица 3.3 – Характеристики матрицы для структуры, представленной на рисунке 3.4

N	N_c	Плотность матрицы **A**, %	Плотность матрицы **M**, %	Число обусловленности
1600	1200	99,23	100	$4{,}49 \cdot 10^3$

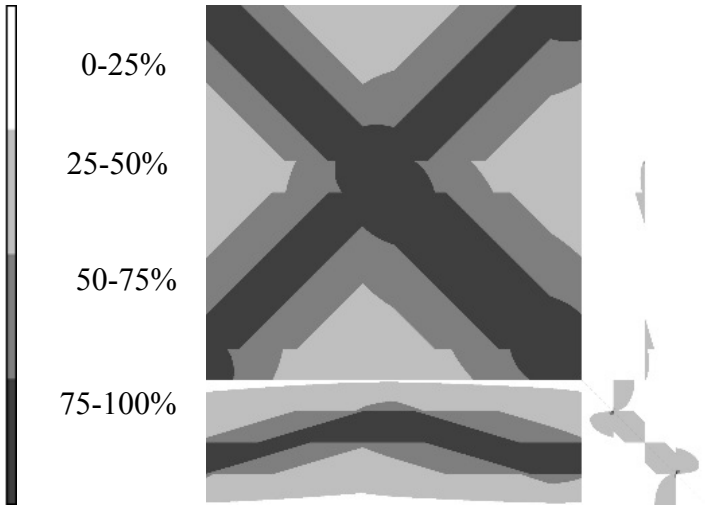


Рисунок 3.7 – Портрет матрицы при $N = 1600$
($h = 12$ мкм, $w = 18$ мкм, $t = 6$ мкм)

Портреты матрицы $\Delta\mathbf{A}_k$ ($k = 100$) при изменении h, w, t показаны на рисунке 3.8. Как видно, изменение каждого размера имеет свою специфику.

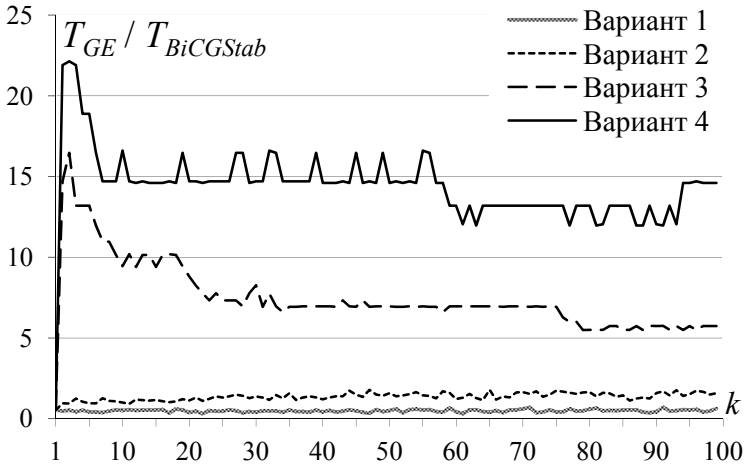
а *б* *в*

Рисунок 3.8 – Портрет матрицы $\Delta\mathbf{A}_k$ ($k = 100$)
при изменении размеров h (*а*); w (*б*); t (*в*)

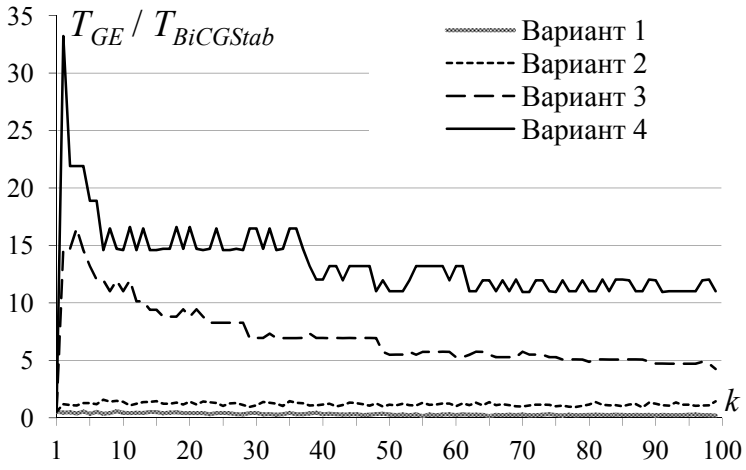
Отношение времени для решения k-й СЛАУ методом Гаусса T_{GE} ко времени ее решения методом BiCGStab $T_{BiCGStab}$ в зависимости от k для вариантов 1–4 приведено на рисунке 3.9.

В проведенных тестах вариант 4 показал максимальное ускорение. При небольших изменениях (в пределах 100 %) размеров можно получить довольно большое ускорение (10–30 раз). Снижение ускорения к концу многократного решения СЛАУ связано

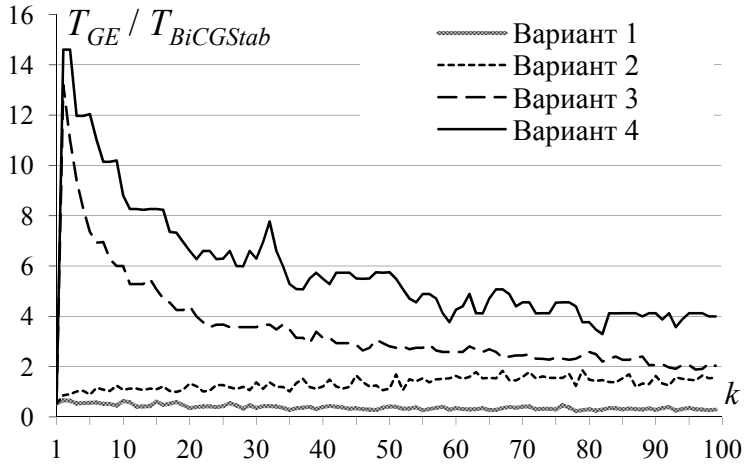
с ростом количества итераций (см. рисунок 3.5) из-за значительного изменения (до 1767 %) размеров.



а



б



в

Рисунок 3.9 – Отношение времени решения k-й СЛАУ методом Гаусса ко времени ее решения методом BiCGStab в зависимости от k для вариантов 1–4 при изменении h (*а*); w (*б*); t (*в*)

Отношения общего времени решения методом Гаусса ко времени решения итерационным методом для 100 СЛАУ приведены в таблице 3.4.

Таблица 3.4 – Ускорение решения 100 СЛАУ

Изменяемый параметр	Вариант 1	Вариант 2	Вариант 3	Вариант 4
h	0,48	1,32	6,49	11,77
w	0,31	1,15	5,87	10,98
t	0,37	1,28	2,87	4,92

Из таблицы 3.4 видно, что лучшие результаты имеет вариант 4 с совместным использованием двух способов ускорения. Вариант 1 показал замедление решения, поскольку в нем не использованы способы ускорения. Для варианта 4 при изменении *t* ускорение меньше, чем при изменении *h* и *w*. Это объясняется ростом количества итераций из-за накопления изменений значений элементов в матрице. При этом используемые способы ускорения недостаточно эффективны.

3.2. Выбор критерия переформирования предобусловливателя

3.2.1. Увеличение числа итераций выше заданного порога

Условия переформирования предобусловливателя позволяют ускорить многократное решение СЛАУ с изменяющейся матрицей итерационным методом [134, 135].

Выше показано, что эффективность многократного решения СЛАУ итерационным методом снижается, если увеличивается разница между первой и текущей матрицами СЛАУ. Это ведет к росту числа итераций и замедляет решение итерационным методом. Корректировка предобусловливателя при неявном предобусловливании неэффективна, поэтому такой подход далее не исследовался. Вместо корректировки предлагается переформировывать матрицу **M** для получения ускорения, когда при решении текущей СЛАУ не обеспечивается быстрая сходимость метода.

Для формулировки условия переформирования матрицы предобусловливания можно воспользоваться различными критериями. Первоначально будем использовать переформирование, когда текущее число итераций N_{it} становится выше заданного порога N_{it}^{MAX}. Приведем разработанный алгоритм итерационного решения m СЛАУ с использованием переформирования матрицы **M** по значению порога числа итераций N_{it}^{MAX}.

Алгоритм 3.2 – Многократное решение СЛАУ с переформированием матрицы предобусловливания по порогу числа итераций

1 Вычислить матрицу **M** из матрицы $\mathbf{A}_1$
2 Положить $N_{it} = 0$
3 Для k от 1 до m
4 Если $N_{it} > N_{it}^{\mathrm{MAX}}$ и $k > 1$, то
5 вычислить матрицу **M** из матрицы $\mathbf{A}_k$
6 Итерационно вычислить $\mathbf{x}_k$ из уравнения $\mathbf{MA}_k\mathbf{x}_k = \mathbf{Mb}$ с заданной точностью
7 Сохранить число итераций в N_{it}
8 Увеличить k

В данном алгоритме в качестве способа формирования матрицы предобусловливания использовано ILU(0)-разложение. Предфильтрация не применялась, поскольку при большом числе СЛАУ алгоритм многократного решения эффективен при нулевом допуске обнуления. Вначале (строка 1) происходит формирование матрицы предобусловливания **M** из матрицы первой СЛАУ. Если текущее число итераций превышает порог (строка 4), то происходит переформирование матрицы **M** (строка 5) и дальнейшее ее использование при решении следующей системы.

Рассмотрим влияние порога числа итераций на решение. Если порог слишком низок, то алгоритм будет вызывать много переформирований (чтобы поддерживать низкое число итераций), следовательно, общие затраты времени будут большими. При слишком высоком пороге переформирований будет мало, но при этом число итераций при решении СЛАУ будет большим, следовательно, общие затраты времени также будут большими. Из при-

веденных предположений следует, что существует оптимальный порог числа итераций, при котором общие затраты времени минимальны. Для проверки этой гипотезы и для первого исследования предложенного алгоритма проведен вычислительный эксперимент.

Использовался персональный компьютер, параметры которого указаны в п. 3.1.2. Исследовалась структура из одного проводника на диэлектрической подложке над идеально проводящей плоскостью (см. рисунок 3.4) в системе TALGAT. Вычислялось 100 емкостных матриц, полученных путем изменения высоты проводника t в диапазоне 6–106 мкм (на 1767 %). Число сегментов на границах структуры не менялось для сохранения порядка (1600) матриц СЛАУ. Анализ матриц приведен в п. 3.1.2. В качестве итерационного метода использовался метод BiCGStab. Итерации продолжались, пока относительная норма вектора невязки была больше 10^{-8}. В качестве начального приближения использовалось решение предыдущей системы (для первой использовался единичный вектор).

Были проведены вычисления с различными значениями порога переформирования. На рисунке 3.10 показано изменение числа итераций при минимальном $\left(N_{it}^{\mathrm{MAX}}=2\right)$ и максимальном $\left(N_{it}^{\mathrm{MAX}}=13\right)$ значениях порога.

Из рисунка 3.10 видно, что значение порога сильно влияет на процесс многократного решения. Так, при минимальном значении порога число переформирований велико (12 — по числу минимумов на нижнем графике), а при максимальном оно равно нулю. График для $N_{it}^{\mathrm{MAX}}=2$ показателен частотой переформирований: она максимальна в начале решения, но уменьшается с ростом k, тем самым отражая способность каждой вновь сформированной матрицы предобусловливания обеспечить решение с изменяющимися матрицами СЛАУ не более чем за 2 итерации. Для более детального анализа в таблице 3.5 приведены характеристики решения для каждого значения N_{it}^{MAX}.

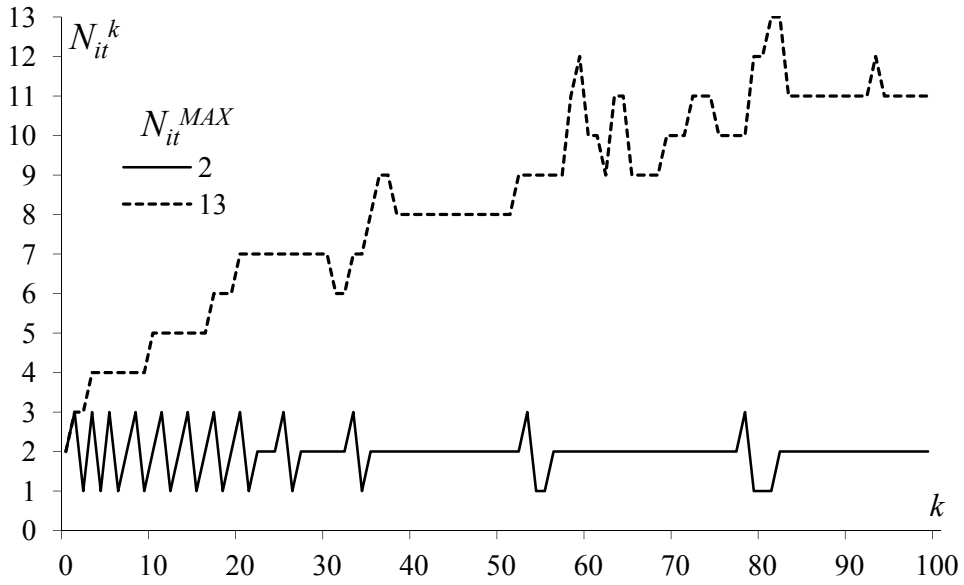


Рисунок 3.10 – Зависимости числа итераций при решении k-й СЛАУ от k при $N_{it}^{\mathrm{MAX}} = 2$ и $N_{it}^{\mathrm{MAX}} = 13$

Таблица 3.5 – Характеристики многократного решения СЛАУ с переформированием матрицы предобусловливания по порогу числа итераций при различных его значениях

Значение порога переформирования матрицы **M** (N_{it}^{MAX})	Число переформирований матрицы **M**	Сумма итераций решения всех СЛАУ	Время решения всех СЛАУ, мс	Ускорение относительно алгоритма без переформирования матрицы **M**
2	12	196	53673	0,77
3	5	249	30701	1,35
4	2	307	24039	1,72
5	2	415	29504	1,40
6	1	416	26040	1,59
7	1	387	24571	1,68
8	1	389	24978	1,66
9	1	487	28903	1,43
10	1	487	29018	1,43
11	1	496	29303	1,41
12	1	666	37122	1,11
13	0	825	41364	1,00

Как следует из таблицы, при минимальном значении порога происходит много переформирований, которые увеличивают об-

щее время решения, поскольку время одного переформирования велико (около 3500 мс). Среднее число требуемых итераций на одно решение очень мало (около 2), поскольку ограничено порогом. Поэтому вклад итераций в общее время решения мал, а основная часть времени расходуется на переформирования. Примечательно, что они избыточны, поскольку увеличивают время решения относительно алгоритма без переформирований (ускорение 0,77 при $N_{it}^{\mathrm{MAX}} = 2$). При максимальном значении порога переформирований нет (ускорение 1,00 при $N_{it}^{\mathrm{MAX}} = 13$) и общие затраты времени определяются затратами на итерации. Но эти затраты значительны, несмотря на малое время (около 50 мс) одной итерации, поскольку среднее число итераций увеличилось в 4 раза. Из анализа данных для промежуточных значений порога видно, что существуют его оптимумы: глобальный (4), при котором достигается максимальное ускорение (1,72), и локальный (7) с несколько меньшим ускорением (1,68). Такие ускорения представляются существенными, если учесть, что они получены лишь за счет переформирований матрицы предобусловливания (дополнительно к ускорениям за счет первого предобусловливания, а также использования решения предыдущей СЛАУ в качестве начального приближения для последующей).

Однако у данного подхода есть недостаток, который не позволяет широко использовать его на практике: определить значение оптимального порога до начала решения не представляется возможным. Поэтому актуален поиск условий переформирования, не обладающих таким недостатком.

3.2.2. Среднее арифметическое время решения

В алгоритмах, используемых далее, в качестве способа формирования матрицы предобусловливания используется LU-разложение, так как предфильтрация отсутствует (см. п. 3.1.1). Поскольку матрицы являются плотными, то нет необходимости применять ILU(0), потому что при таких условиях ILU(0) полностью повторяет LU.

Эффективность итерационного метода при многократном решении СЛАУ определяется суммарным временем решения всех СЛАУ T_Σ: чем меньше это время, тем эффективней метод [132]. Без переформирования матрицы **M** (аналогично (3.1))

$$T_\Sigma = T_{PR} + \sum_{k=1}^{m} T_k, \tag{3.4}$$

где T_{PR} — время формирования матрицы предобусловливания **M** из матрицы первой СЛАУ; T_k — время решения k-й СЛАУ; m — количество СЛАУ.

Используя выражение (3.4), рассмотрим изменение среднего арифметического времени решения k СЛАУ:

$$\overline{T}(k) = \frac{T_\Sigma(k)}{k} = \frac{1}{k}\left(T_{PR} + \sum_{j=1}^{k} T_j \right). \tag{3.5}$$

Для удобства введем дополнительную величину s_k:

$$s_k = kT_{k+1} - \sum_{j=1}^{k} T_j. \tag{3.6}$$

Теорема 1. Если s_k монотонно возрастает при $k = 1, 2, \ldots, k_*, \ldots$ и выполнены условия $s_1 < T_{PR}$ и $s_{k_*} < T_{PR} \le s_{k_*+1}$ для некоторого $k_* \ge 1$, для этого k_* достигается единственный минимум функции $\overline{T}(k)$.

Доказательство. Рассмотрим разность $\overline{T}(k+1) - \overline{T}(k)$, тогда

$$\overline{T}(k+1) - \overline{T}(k) = \frac{1}{k+1}\left(T_{PR} + \sum\nolimits_{j=1}^{k+1} T_j\right) - \frac{1}{k}\left(T_{PR} + \sum\nolimits_{j=1}^{k} T_j\right) =$$

$$= \frac{1}{k(k+1)}\left(kT_{PR} - (k+1)T_{PR} + k\sum\nolimits_{j=1}^{k+1} T_j - (k+1)\sum\nolimits_{j=1}^{k} T_j\right) =$$

$$= \frac{1}{k(k+1)}\left(-T_{PR} + kT_{k+1} - \sum\nolimits_{j=1}^{k} T_j\right) = \frac{1}{k(k+1)}\left(-T_{PR} + s_k\right).$$

Очевидно, что функция $\overline{T}(k)$ убывает, если $s_k < T_{PR}$, в противном случае ($s_k > T_{PR}$) функция $\overline{T}_k(k)$ возрастает. Для наличия минимума функции $\overline{T}_k(k)$ достаточно выполнения условия $s_1 < T_{PR}$ и для некоторого конечного k_* $s_{k*} < T_{PR} \le s_{k*+1}$. Теорема доказана.

Теорема 2. Если при $k = 1, 2, \ldots$ выполняются условия $T_k = t$ и $t < T_{PR}$, тогда функция $\overline{T}(k)$ убывает и стремится к t.

Доказательство. Пусть $T_k = t$, тогда согласно выражению (3.6) $s_k = 0$, а

$$\overline{T}(k+1) - \overline{T}(k) = \frac{1}{k+1}\left(-T_{PR}\right) < 0.$$

Из этого следует, что функция $\overline{T}(k)$ убывает для всех k и согласно равенству (3.5)

$$\lim_{k\to\infty} \overline{T}(k) = \lim_{k\to\infty}\left(\frac{1}{k}\left(T_{PR} + kt\right)\right) = t.$$

Теорема доказана.

Условия, указанные в теореме 1, выполняются при многократном решении СЛАУ. То есть для монотонного роста s_k достаточно увеличения времени решения k-й СЛАУ T_k, что и наблюдается (см. п. 3.1.1). Как правило, решение первой СЛАУ происходит значительно быстрее, чем время формирования матрицы предобусловливателя, т.е. выполняется условие $s_1 < T_{PR}$. Из этого следует, что при многократном решении СЛАУ будет существовать единственный минимум функции $\overline{T}_k$, что позволяет использовать его как условие для переформирования предобусловливателя с помощью LU-разложения (т.е. при выполнении условия $\overline{T}(k-1) < \overline{T}_k$ или $\frac{T_\Sigma(k-1)}{k-1} < \frac{T_\Sigma(k)}{k}$). Из теоремы 2 следует, что при постоянном T_k, что возможно только при постоянном и низком числе итераций, функция $\overline{T}(k)$ будет постоянно убывать, а значит, переформирование предобусловливателя не будет

происходить, что и не требуется при низком числе итераций. С учетом вышеизложенного разработан алгоритм 3.3.

Алгоритм 3.3 – Многократное решение СЛАУ с переформированием матрицы предобусловливания по увеличению среднего арифметического времени решения

1 Вычислить матрицу предобусловливания **M** из матрицы $\mathbf{A}_1$
2 Сохранить время решения в T_{PR}
3 Найти $\mathbf{x}_1$ из уравнения $\mathbf{MA}_1\mathbf{x}_1 = \mathbf{Mb}$ с заданной точностью *Tol*
4 Сохранить время решения в T_1
5 $T_\Sigma = T_{PR} + T_1$
6 Для k от 2 до m
7 Найти $\mathbf{x}_k$ из уравнения $\mathbf{MA}_k\mathbf{x}_k = \mathbf{Mb}$ с заданной точностью *Tol*
8 Сохранить время решения в T_k
9 Если $T_\Sigma/(k–1) < (T_\Sigma + T_k)/k$ и $k \neq m$
10 Вычислить матрицу предобусловливания **M** из матрицы $\mathbf{A}_k$
11 Сохранить время вычисления в T_{PR}
12 $T_\Sigma = T_\Sigma + T_{PR}$
13 $T_\Sigma = T_\Sigma + T_k$
14 Увеличить k

В данном алгоритме в качестве способа формирования матрицы предобусловливания использовано LU-разложение. В строке 7 выполняется решение СЛАУ итерационным методом. Далее проверяется условие для переформирования матрицы предобусловливания (строка 9). Если среднее арифметическое значение времени решения на шаге k $\left(\left(T_\Sigma + T_k\right)/k\right)$ больше, чем на шаге k–1 $\left(T_\Sigma/\left(k-1\right)\right)$, то происходит переформирование матрицы предобусловливания, которое реализуется в строке 10.

Для подтверждения сделанного предположения выполнен простой эксперимент по вычислению емкости проводника на диэлектрической подложке над идеально проводящей плоскостью (см. рисунок 3.4). Использовался персональный компьютер (без распараллеливания, т.е. работало одно ядро процессора) с параметрами: платформа Intel(R) Core (TM) i7; частота процессора 2,80 ГГц; объем ОЗУ 12 Гбайт; число ядер — 8; операционная система Windows 8x64. Применялся итерационный метод BiCGStab. Матрицы получены в системе TALGAT. Число сегмен-

тов на каждом отрезке структуры не менялось для постоянства порядка матриц СЛАУ ($N = 1600$). В качестве способа формирования матрицы предобусловливания использовалось LU-разложение. Для всех СЛАУ вектор **b** — единичный. Как начальное приближение выбрано решение предыдущей системы, для первой использовался единичный вектор. Итерации продолжались, пока относительная норма вектора невязки была больше 10^{-8}. Зависимости T_k и $\overline{T}_k$ приведены на рисунке 3.11, из которого видно наличие минимума в поведении среднего арифметического значения (при $k = 52$), что подтверждает сделанное предположение.

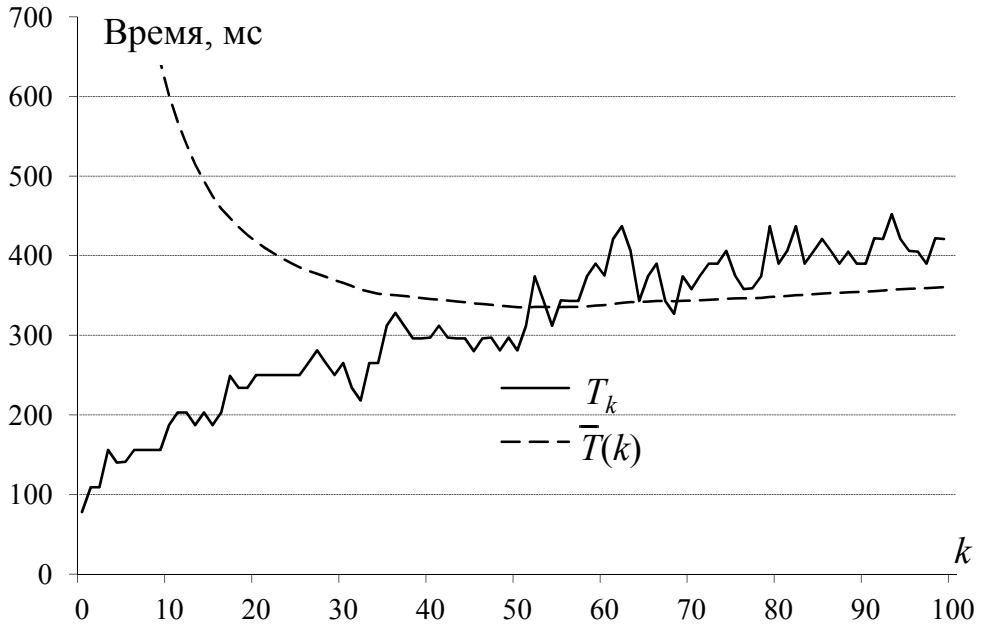


Рисунок 3.11 – Зависимости времени решения k-й СЛАУ T_k и среднего арифметического $\overline{T}(k)$ от k

Далее для тестирования алгоритма сформировано 100 матриц для двух рассмотренных структур. Для структуры 1 (см. рисунок 3.4) матрицы порядка $N = 1600$ получены путем изменения высоты проводника t в диапазоне 6, 7, … 106 мкм. Для структуры 2 (рисунок 3.12), представляющей собой модальный фильтр [136], матрицы порядка $N = 2001$ получены путем изменения зазоров s в диапазоне 100, 101, … 200 мкм. Учащением сегментации получены матрицы порядков 3200 и 3001.

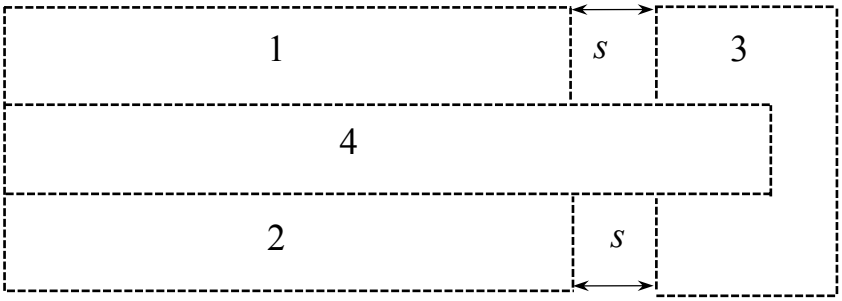


Рисунок 3.12 – Вид поперечного сечения структуры 2 (модальный фильтр): проводники 1, 2 и 3, между которыми помещен диэлектрик 4

Полученные матрицы проанализированы, результаты анализа представлены в таблице 3.6. На рисунке 3.13 приведены портреты матриц при $k = 1$.

Таблица 3.6 – Характеристики используемых матриц для структуры 1 (см. рисунок 3.4) и структуры 2 (см. рисунок 3.12)

Структура	N	N_c	Плотность матрицы **A**, %	Плотность матрицы **M**, %	Число обусловленности
1	1600	1200	99,23	100	$4{,}49 \cdot 10^3$
	3200	2400	99,22	100	$9{,}07 \cdot 10^3$
2	2001	1700	97,74	100	$2{,}93 \cdot 10^8$
	3001	2550	97,74	100	$6{,}39 \cdot 10^8$

Достигнутые ускорения решения 100 СЛАУ алгоритмом 3.3 для структур 1 и 2 по отношению к алгоритму без переформирования предобусловливателя показаны в таблице 3.7. Здесь же приведены результаты, полученные при переформировании предобусловливателя с помощью задания оптимального порога числа итераций (см. п. 3.2.1).

Максимальное ускорение обеспечивает алгоритм с заданием оптимального порога числа итераций. Однако этот алгоритм требует определения значения этого порога путем пошагового поиска. Алгоритм переформирования по увеличению $\overline{T}(k)$ показал ускорения, близкие к оптимальным. Отличия составляют не более 7,6 % (для структуры 2 при $N = 3001$).

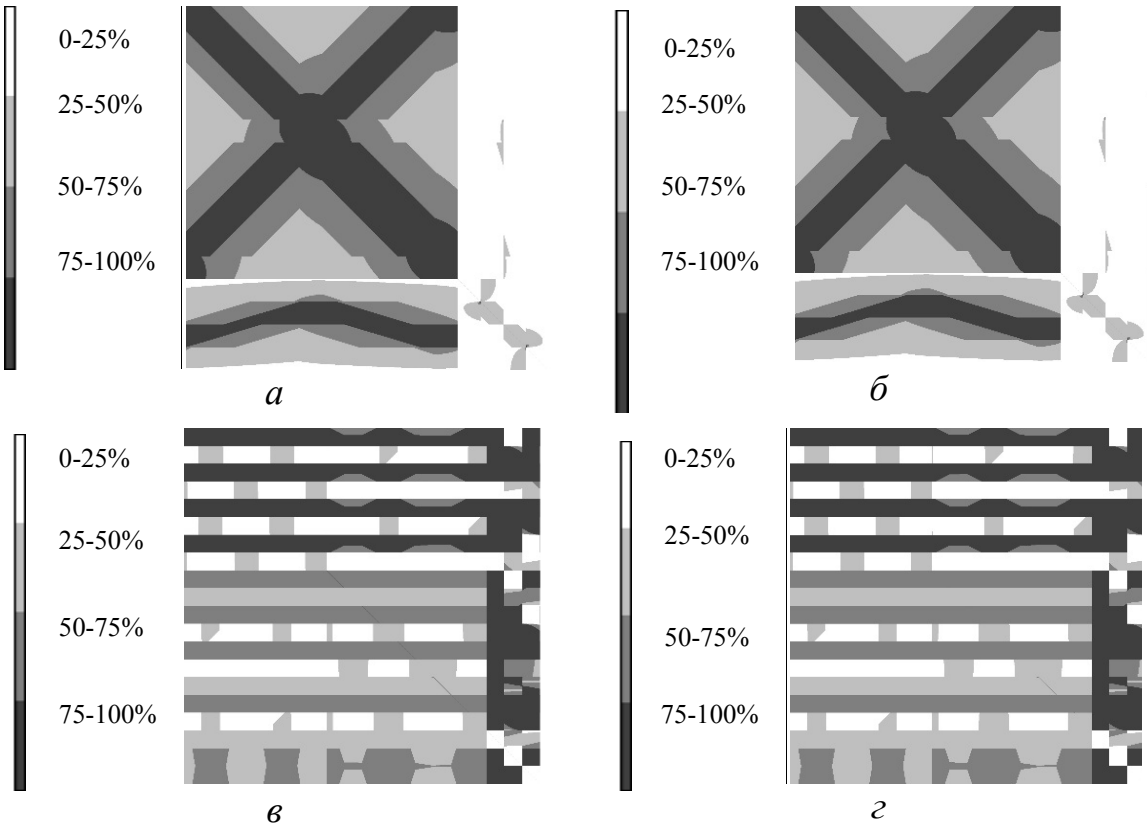
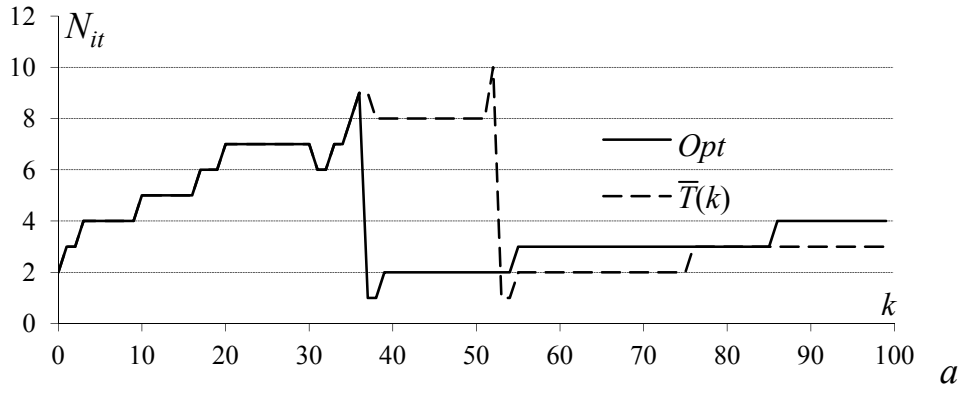


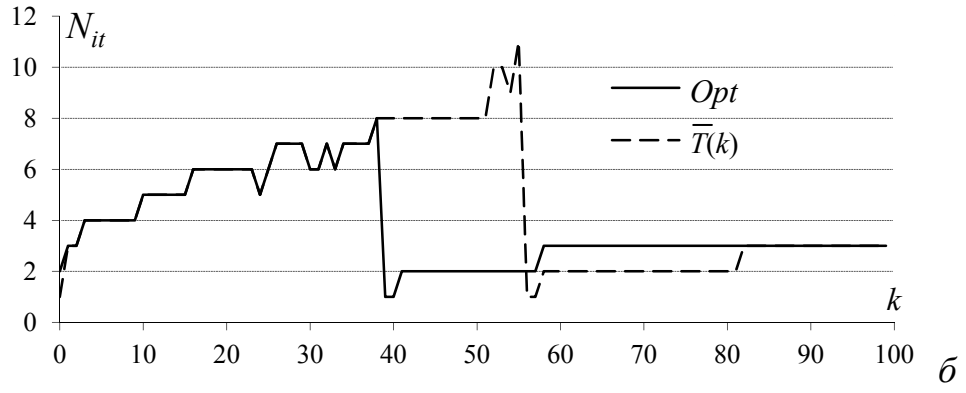
Рисунок 3.13 – Портреты матриц при разных N: 1600 (*а*); 3200 (*б*); 2001 (*в*); 3001 (*г*)

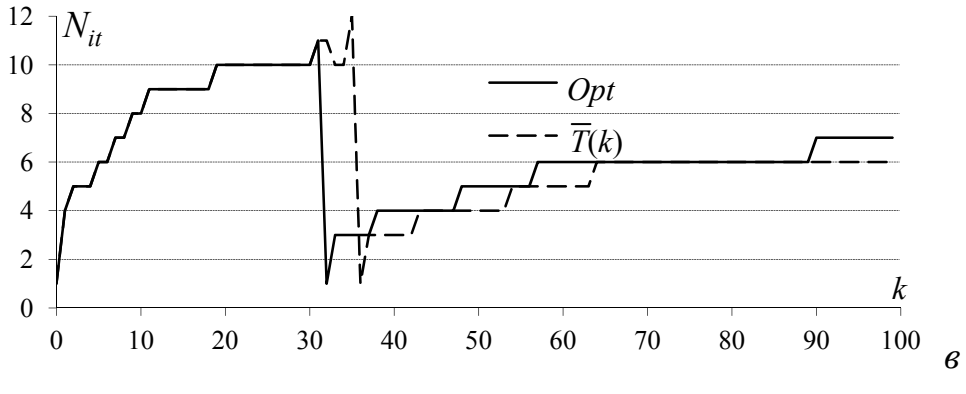
Таблица 3.7 – Ускорение решения 100 СЛАУ методом BiCGStab с переформированием предобусловливателя

Алгоритм	Ускорение			
	Структура 1		Структура 2	
	$N = 1600$	$N = 3200$	$N = 2001$	$N = 3001$
По оптимальному порогу числа итераций (N_{it}^{Opt})	1,51 (8)	1,24 (8)	1,62 (10)	1,58 (10)
По увеличению среднего арифметического времени	1,51	1,15	1,60	1,46

На рисунке 3.14 для структур 1 и 2 показано, как меняется число итераций при решении k-й СЛАУ с использованием обоих алгоритмов: по увеличению $\overline{T}(k)$ и по оптимальному порогу числа итераций (*Opt*).







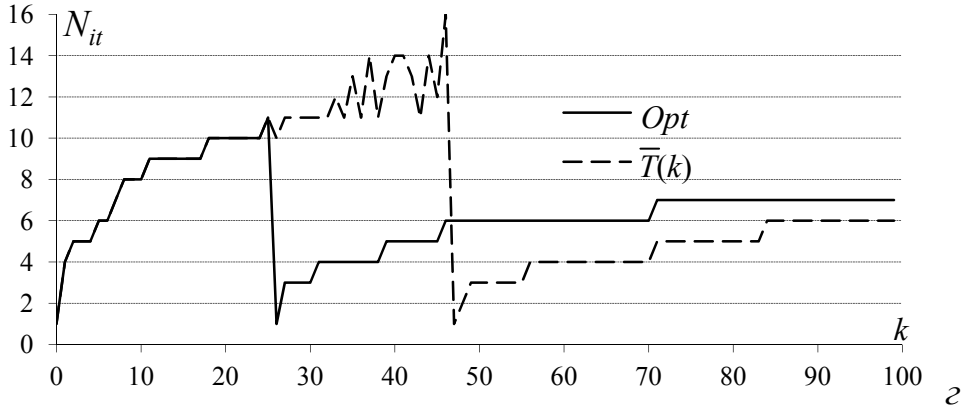


Рисунок 3.14 – Число требуемых итераций в зависимости от k при использовании алгоритма с переформированием предобусловливателя по увеличению $\overline{T}(k)$ и по заданному оптимальному порогу числа итераций (*Opt*) для структуры 1 при $N = 1600$ (*а*), 3200 (*б*) и структуры 2 при $N = 2001$ (*в*), 3001 (*г*)

Видно, что при более позднем переформировании предобусловливателя ускорение уменьшается. Например, для структуры 1 (рисунок 3.14,*б*) алгоритм по увеличению $\overline{T}(k)$ (переформирование при $k = 56$) дает ускорение 1,15, а алгоритм по оптимальному порогу — 1,24 (переформирование при $k = 39$). Таким образом, алгоритм по увеличению $\overline{T}(k)$ адаптивно определяет момент переформирования матрицы предобусловливания при увеличении времени итерационного процесса, тем самым минимизируя время решения всех СЛАУ (независимо от загрузки центрального процессора рабочей станции посторонними задачами).

3.2.3. Средняя арифметическая сложность решения

Использование среднего арифметического времени решения имеет недостаток, обусловленный трудностью точного измерения времени [137]. Поэтому предложено перейти от измерений времени к расчету сложности алгоритмов [138].

Общую сложность решения (без переформирования предобусловливателя) m СЛАУ F_{Σ} можно выразить через среднюю арифметическую сложность решения одной СЛАУ $\overline{F}: F_{\Sigma} = m\overline{F}$. Тогда вычислительные затраты многократного решения СЛАУ можно отслеживать, контролируя значение текущей средней арифметической сложности $\overline{F_k}$ решения k СЛАУ: $\overline{F}_k = F_{\Sigma k}/k$. Для определения сложности будем использовать арифметическую сложность Q [137] и O-нотацию.

Данные подходы применимы в любом итерационном методе. Для вычислительного эксперимента выбраны два итерационных метода — BiCGStab и CGS [86]. Для получения матрицы предобусловливателя **M** использовалось LU-разложение.

При расчете сложности алгоритма BiCGStab учитывалось наличие двух возможных условий выхода из итерационного процесса (алгоритм 1.2, строки 17 и 24). Полученные оценки сложности для LU-разложения, BiCGStab и CGS приведены в таблице 3.8. Подробный расчет сложности алгоритмов приведен в разд. 4.

Таблица 3.8 – Оценки сложности алгоритмов

Алгоритм	Арифметическая сложность Q	O-нотация
LU-разложение	$f_{LU}(N) = (10N^3 - 3N^2 + 16N - 18)/6$	$f_{LU}(N) = N^3/6$
BiCGStab, условие 1 (выход в строке 17)	$f(N, N_{it}) = 5N^2 + 13N + 3 + N_{it}(10N^2 + 31N + 16) + (N_{it} - 1)(10N^2 + 33N + 11)$	$f(N, N_{it}) = 4N^2 + 6N + 4(N_{it} - 1)(N^2 + 2N)$
BiCGStab, условие 2 (выход в строке 24)	$f(N, N_{it}) = 5N^2 + 8N + 2 + N_{it}(20N^2 + 64N + 37)$	$f(N, N_{it}) = 2N^2 + 2N + 4N_{it}(N^2 + 2N)$
CGS	$f(N, N_{it}) = 10N^2 + 16N + 1 + N_{it}(20N^2 + 43N + 15) + (N_{it} - 1)(10N + 4)$	$f(N, N_{it}) = 2N^2 + N_{it}(8N^2 + 5N)$

Для верификации оценок проведен вычислительный эксперимент. Использовался персональный компьютер, параметры которого указаны в п. 3.2.2. Матрицы получены в системе TALGAT. Для тестирования сформировано по 100 матриц для каждого рассматриваемого случая. Исследованы три структуры. Для структуры 1 (см. рисунок 3.4) матрицы с $N = 1600$ получены путем изменения высоты проводника t в диапазоне 6, 7, …, 105 мкм. Для структуры 2 (см. рисунок 3.12) матрицы с $N = 2001$ получены путем изменения зазоров s в диапазоне 100, 101, …, 199 мкм. Для структуры 3 (рисунок 3.15), представляющей собой модальный фильтр [139], матрицы с $N = 1709$ получены путем изменения зазоров s в диапазоне 16,9; 16,8; …; 7,1 мкм. Далее учащением сегментации получены матрицы порядков 3200, 3001 и 3109 соответственно. Для всех СЛАУ вектор **b** — единичный.

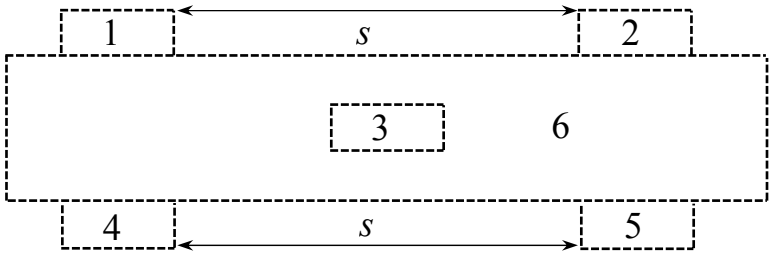


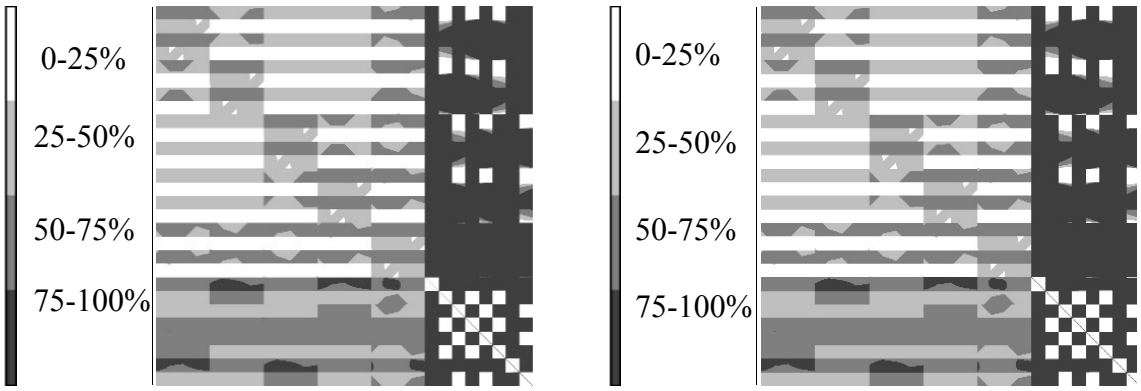
Рисунок 3.15 – Вид поперечного сечения исследуемой структуры – модальный фильтр (1–5 – проводники, 6 – диэлектрик)

Характеристики матриц для структур 1 и 2 приведены в таблице 3.6. Для структуры 3 результаты анализа сведены в таблицу 3.9. На рисунке 3.16 показаны портреты матриц структуры 3.

Таблица 3.9 – Характеристики матриц для структуры 3 (рисунок 3.15)

N	N_c	Плотность матрицы **A**, %	Плотность матрицы **M**, %	Число обусловленности
1709	1220	95,9	100	$2{,}57 \cdot 10^7$
3109	2220	95,9	100	$8{,}29 \cdot 10^7$

Время вычисления и оценки сложности алгоритма LU-разложения, нормированные по максимальным значениям, представлены на рисунке 3.17. Видно, что характер графиков одинаков.



а *б*

Рисунок 3.16 – Портреты матриц для структуры 3 при разных N: 1709 (*а*); 3109 (*б*)

На рисунке 3.18 приведены отклонения нормированных значений оценок O, Q от нормированного измеренного времени T, полученные по формулам

$$\Delta O = \left| \frac{T - O}{T} \right| \cdot 100\ \%, \tag{3.7}$$

$$\Delta Q = \left| \frac{T - Q}{T} \right| \cdot 100\ \%. \tag{3.8}$$

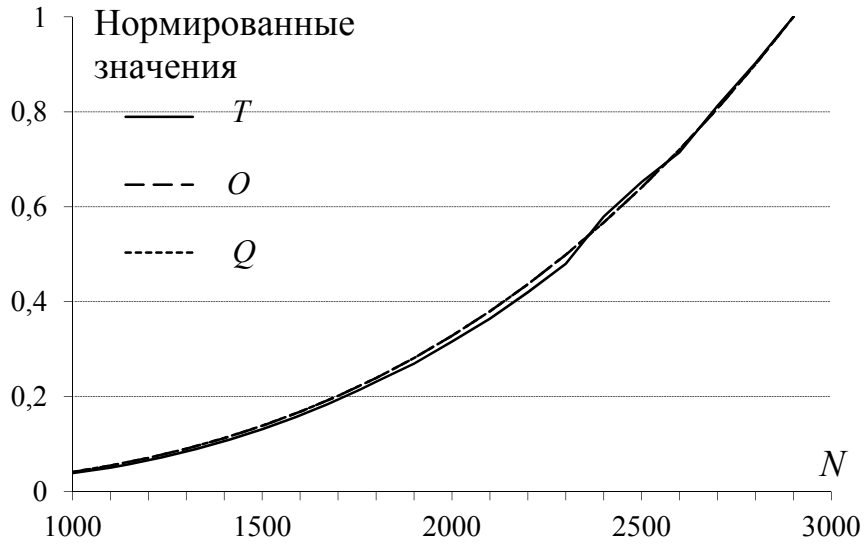


Рисунок 3.17 – Нормированные значения времени вычисления T, оценки сложности с помощью O-нотации и арифметической сложности Q LU-разложения в зависимости от N

Видно, что отклонения не превышают 9 % и уменьшаются с ростом N. Таким образом, оценки сложности LU-разложения получены верно.

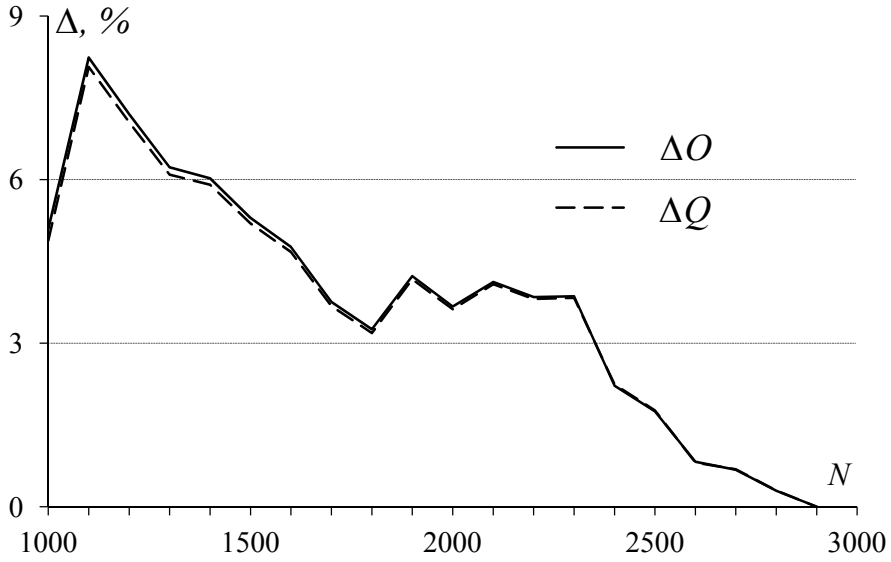


Рисунок 3.18 – Отклонения оценок сложности ΔO, ΔQ LU-разложения в зависимости от N

Для аналогичного сравнения оценок сложности алгоритмов методов BiCGStab и CGS использовано 100 СЛАУ. Взята структура 1 (см. рисунок 3.4) и матрицы порядка $N = 1600$. Итерации

продолжались, пока относительная норма вектора невязки была больше 10^{-8}. Для более корректной оценки искусственно увеличено время решения СЛАУ путем использования предобусловливателя Якоби, в котором учитываются только диагональные элементы матриц. За счет этого увеличилось количество итераций при решении СЛАУ и, как следствие, измеряемое время. Время решения *k*-й СЛАУ методом BiCGStab (без переформирования предобусловливателя) и число требуемых итераций представлены на рисунке 3.19. Нормированные (по максимальному значению) время решения и оценки сложности метода BiCGStab приведены на рисунке 3.20.

Видно, что графики практически совпадают. Для более точного сравнения на рисунке 3.21 приведены отклонения нормированных значений оценок от нормированного измеренного времени (согласно выражениям (3.7) и (3.8)).

Видно, что отклонения малы (максимальное около 4 %), что подтверждает верность оценок. Таким образом, аналитические оценки подтверждаются вычислительным экспериментом.

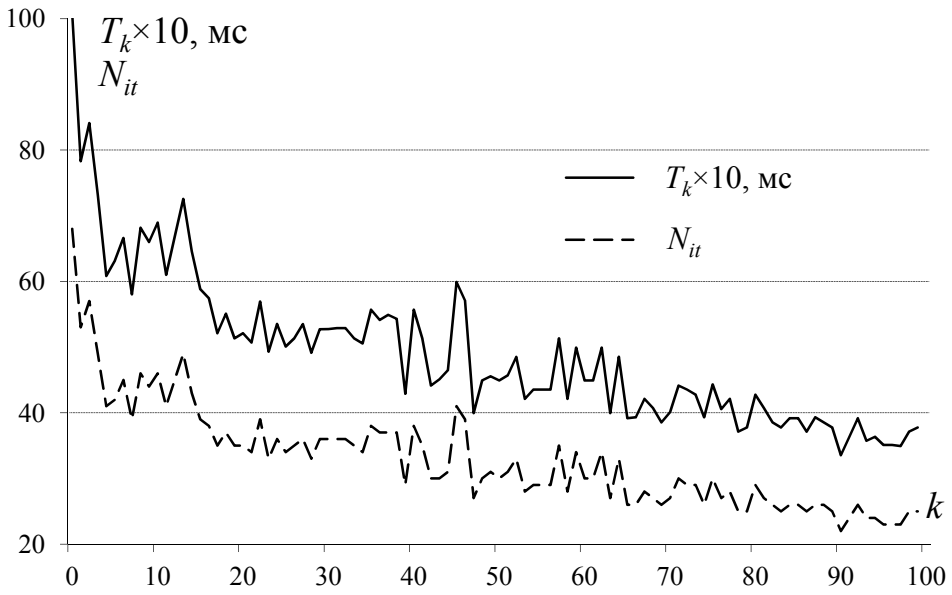


Рисунок 3.19 – Число итераций N_{it} и время решения T_k *k*-й СЛАУ методом BiCGStab с предобусловливателем Якоби в зависимости от *k*

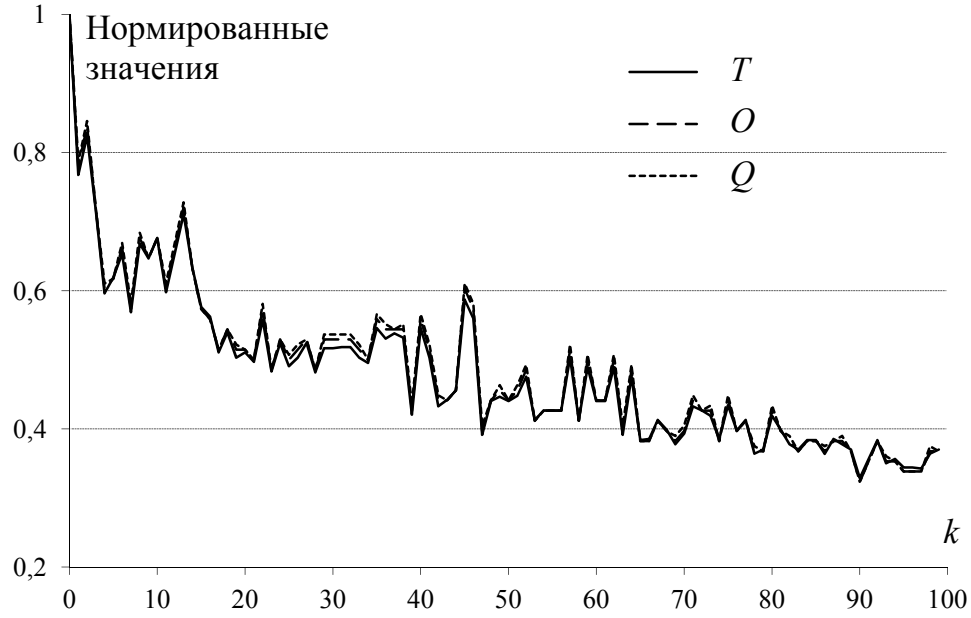


Рисунок 3.20 – Нормированные значения времени решения k-й СЛАУ T и нормированные оценки с помощью O-нотации и арифметической сложности Q метода BiCGStab с диагональным предобусловливателем в зависимости от k

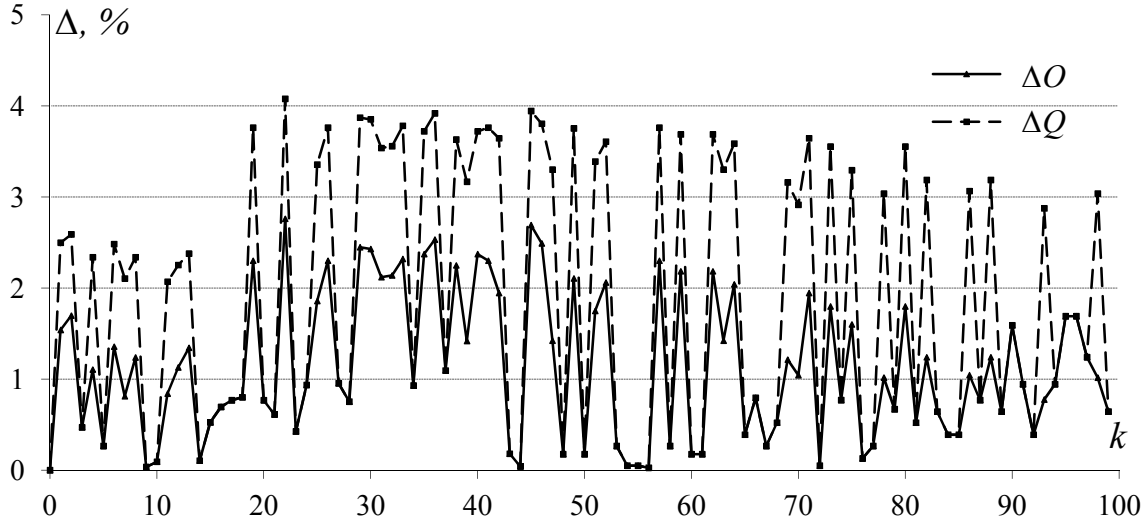


Рисунок 3.21 – Отклонения оценок ΔO, ΔQ для алгоритма BiCGStab в зависимости от k

Далее приведен разработанный алгоритм многократного решения СЛАУ с переформированием предобусловливателя, выполняемым на основе оценок из таблицы 3.8.

Алгоритм 3.4 – Многократное решение СЛАУ с переформированием предобусловливателя при увеличении средней сложности решения k СЛАУ

1 Вычислить матрицу **M** из матрицы $\mathbf{A}_1$
2 $F_\Sigma = f_{LU}(N)$
3 Найти $\mathbf{x}_1$ из уравнения $\mathbf{MA}_1\mathbf{x}_1 = \mathbf{Mb}$ с заданной точностью *Tol*
4 Сохранить количество итераций в N_{it}
5 $F_1 = f(N, N_{it})$
6 $F_\Sigma = F_\Sigma + F_1$
7 Для k от 2 до m
8 Найти $\mathbf{x}_k$ из уравнения $\mathbf{MA}_k\mathbf{x}_k = \mathbf{Mb}$ с заданной точностью *Tol*
9 Сохранить количество итераций в N_{it}
10 $F_k = f(N, N_{it})$
11 Если $F_\Sigma / (k–1) < (F_\Sigma + F_k) / k$ и $k \neq m$
12 Вычислить матрицу **M** из матрицы $\mathbf{A}_k$
13 $F_\Sigma = F_\Sigma + f_{LU}(N)$
14 $F_\Sigma = F_\Sigma + F_k$
15 Увеличить k

В данном алгоритме в качестве способа формирования матрицы предобусловливания использовано LU-разложение. В F_Σ хранится значение общей сложности k решенных СЛАУ для отслеживания изменения средней сложности, при увеличении которой (строка 11) происходит переформирование матрицы предобусловливания **M** (строка 12). В противном случае решение продолжается без переформирования. В F_k хранится значение сложности итерационного процесса (строки 5 и 10). Для каждого итерационного метода F_k определяется согласно таблице 3.8.

Далее выполнен вычислительный эксперимент по многократному решению СЛАУ с переформированием предобусловливателя. Полученные ускорения решения 100 СЛАУ для структур 1, 2 и 3 по отношению к алгоритму без переформирования предобусловливателя (для каждого итерационного метода отдельно) представлены в таблице 3.10. Здесь же приведены результаты, полученные при переформировании предобусловливателя с помощью задания оптимального порога числа итераций (*Opt*).

Таблица 3.10 – Ускорение решения 100 СЛАУ методами BiCGStab и CGS с переформированием предобусловливателя

Структура	N	По оптимальному порогу числа итераций (*Opt*)		По средней арифметической сложности решения (*O*-нотация)		По средней арифметической сложности решения (*Q*)	
		BiCGStab	CGS	BiCGStab	CGS	BiCGStab	CGS
1	1600	1,42 (8)	1,34 (7)	1,20	1,52	1,12	1,40
	3200	1,16 (8)	1,35 (7)	0,92	1,27	0,86	1,01
2	2001	1,62 (10)	1,44 (8)	1,60	0,94	1,52	1,05
	3001	1,58 (10)	1,31 (9)	1,55	0,87	1,44	0,95
3	1709	1,12 (9)	1,18 (8)	1,55	1,57	1,55	1,57
	3109	1,03 (10)	1,00(16)	1,59	1,33	1,59	1,33

Максимальное ускорение обеспечивает алгоритм с переформированием предобусловливателя посредством задания оптимального порога числа итераций (исключение для структуры 1 при $N = 1600$ и использовании метода CGS и для всех тестов структуры 3). Ускорение многократного решения СЛАУ предложенными алгоритмами не постоянно. Например, метод BiCGStab для структуры 1 показал отклонения значений ускорений до 26 %, для структуры 2 — не более 9 %, для структуры 3 — 55 %. При использовании метода CGS для структуры 1 отклонения составили до 28 %, а для структуры 2 и 3 — до 35 и 34 % соответственно. Такие отклонения объясняются тем, что переформирование предобусловливателя происходит в разные моменты. Для детального анализа полученных результатов на рисунке 3.22 показано число итераций при решении *k*-й СЛАУ с использованием приведенных алгоритмов для обеих структур при использовании методов BiCGStab и CGS.

Из анализа графиков можно сделать вывод, что при более позднем переформировании предобусловливателя ускорение уменьшается. Например, для структуры 1 (рисунок 3.22,*в*) при использовании арифметической сложности Q (переформирование при $k_p = 64$) ускорение отсутствует, а при использовании оптимального порога ускорение составляет 1,16 (переформирование при $k_p = 39$).

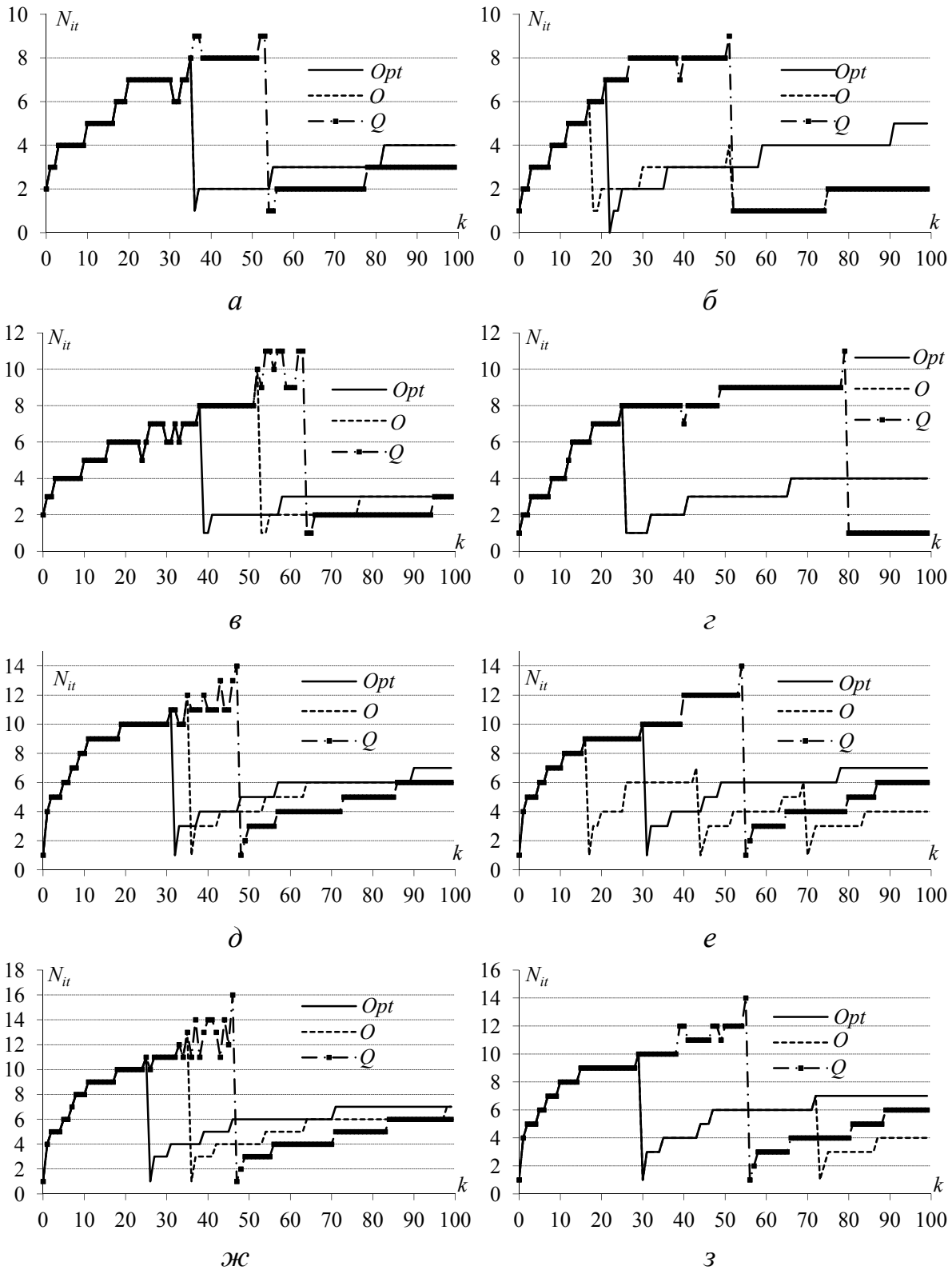


Рисунок 3.22 – Число требуемых итераций алгоритмами с оценкой сложности алгоритма по *O*-нотации (*O*), по арифметической сложности (*Q*) и использующего оптимальный порог числа итераций (*Opt*) при решении методом BiCGStab: для структур 1 при $N = 1600$ (*а*), 3200 (*в*) и 2 при $N = 2001$ (*д*), 3001 (*ж*); CGS: для структур 1 при $N = 1600$ (*б*), 3200 (*г*) и 2 при $N = 2001$ (*е*), 3001 (*з*)

В результате можно сделать вывод, что на скорость решения СЛАУ сильное влияние оказывает момент переформирования, т.е. при каком k_p это происходит. При более позднем переформировании предобусловливателя ускорение уменьшается. Кроме того, замечено, что, чем ближе момент переформирования к m, тем меньшее ускорение будет получено.

3.3. Использование оптимального порядка решения

3.3.1. Алгоритм с выбором оптимального порядка решения СЛАУ

Рассмотрим подход к выбору оптимальной очередности решения СЛАУ [135, 138, 140].

Одним из ресурсов ускорения представляется использование определенной очередности решения СЛАУ. Действительно, порядок решения обычно определяется заданным изменением параметра структуры. Но если общее время решения всех СЛАУ зависит от того, какая именно СЛАУ будет решаться первой, какая — второй и т.д., то существует оптимальная очередность по критерию минимального времени решения. То, что такая зависимость существует, следует из самой сути многократного решения СЛАУ итерационным методом с предобусловливанием. Очевидно, что она определяется двумя факторами: выбором матрицы для вычисления предобусловливателя (то, какая именно из всех матриц будет выбрана, влияет на число итераций, требуемых для решения последующих СЛАУ), а также использованием решения предыдущей СЛАУ в качестве начального приближения текущей (чем ближе начальное приближение окажется к решению, тем меньше потребуется итераций) [132]. При многовариантном анализе используют несколько основных видов изменения параметра: линейное, логарифмическое, с заданными пользователем значениями. При оптимизации изменение может быть случайное, причем в любом направлении. Рассмотрим самое простое, но широко используемое изменение: линейное. При нем изменение очередности решения сводится к тривиальному выбору порядка решения,

т.е. по возрастанию параметра (прямой порядок) или его убыванию (обратный порядок).

Отметим, что линейное изменение параметра вовсе не гарантирует монотонного изменения элементов матрицы СЛАУ или ее нормы, но может быть частым на практике. В любом случае полезен анализ конкретных структур. Далее рассмотрено линейное изменение параметра, что позволяет, меняя только порядок решения (прямой или обратный), выявить оптимальный порядок. В результате разработан общий алгоритм 3.5. В нем в качестве способа формирования матрицы предобусловливания использовано LU-разложение.

Алгоритм 3.5 – Многократное решение m СЛАУ с выбором очередности решения

1 Задать очередность решения СЛАУ
2 Вычислить матрицу предобусловливания **M** из матрицы $\mathbf{A}_1$
3 Для k от 1 до m
4 Найти $\mathbf{x}_k$ из уравнения $\mathbf{MA}_k\mathbf{x}_k = \mathbf{Mb}$ с заданной точностью *Tol*
5 Увеличить k

3.3.2. Вычислительный эксперимент

Для вычислительного эксперимента использовался персональный компьютер, параметры которого указаны в п. 3.2.2. Исследовались итерационные методы BiCGStab и CGS. Рассматривались структура 1 (см. рисунок 3.4), структура 2 (см. рисунок 3.12) и структура 3 (см. рисунок 3.15). Характеристики используемых матриц для структур 1 и 2 приведены в таблице 3.6, а для структуры 3 — в таблице 3.9. В таблице 3.11 представлены полученные ускорения при использовании обратного порядка решения СЛАУ относительно прямого.

Видно, что для всех структур наблюдается ускорение при использовании обратного порядка решения СЛАУ, причем оно получено для всех N и обоих итерационных методов. Ускорение достигается разницей в числе требуемых итераций при решении СЛАУ в прямом (N_{it}^{+}) и обратном (N_{it}^{-}) порядке, выражаемой в

площади фигуры, ограниченной кривыми числа итераций (для метода BiCGStab — рисунок 3.23, для CGS — рисунок 3.24).

Таблица 3.11 – Ускорение решения 100 СЛАУ методами BiCGStab и CGS с переформированием предобусловливателя

Струк-тура	*N*	С использованием обратного порядка решения СЛАУ	
		BiCGStab	CGS
1	1600	1,76	1,73
	3200	1,63	1,66
2	2001	1,71	1,58
	3001	1,84	1,53
3	1709	1,82	1,59
	3109	1,83	1,32

Для всех структур при решении СЛАУ в прямом порядке число итераций больше, чем при решении в обратном порядке. Основная причина этого заключается в разных матрицах СЛАУ, из которых получены предобусловливатели: в прямом порядке предобусловливатель получен из первой СЛАУ, а в обратном — из 100-й.

На число требуемых итераций повлияла и различная степень изменений матрицы СЛАУ в начале (сильная) и конце (слабая) диапазона. Результаты эксперимента подтверждают предположение о влиянии очередности решения СЛАУ на его эффективность.

Таким образом, оценено влияние очередности многократного решения СЛАУ итерационным методом на общее время решения всех СЛАУ. Полученные ускорения вычислений обратного порядка (до 1,84 раза) относительно вычислений в прямом порядке показывают перспективность контроля очередности решения СЛАУ.

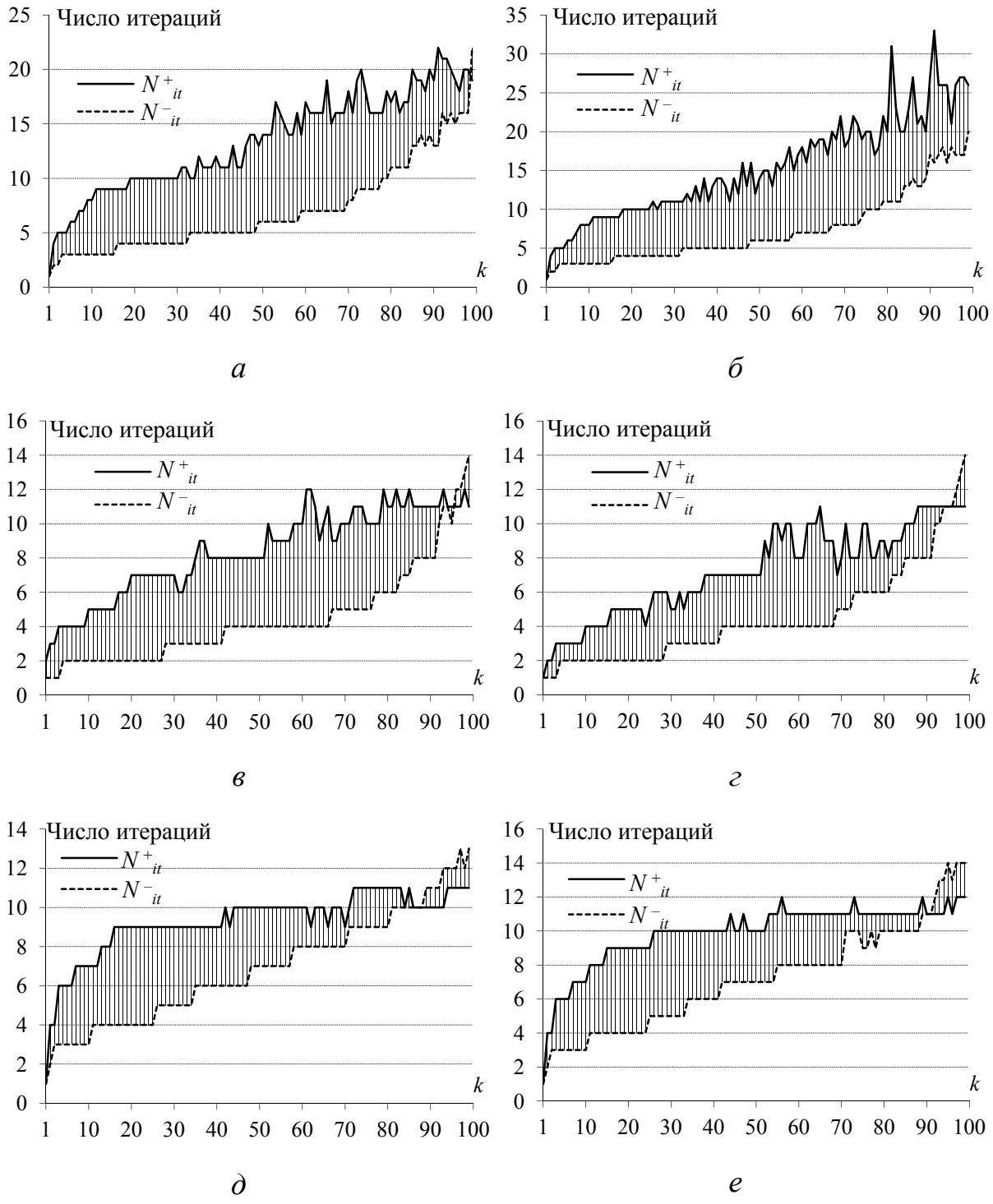


Рисунок 3.23 – Число итераций при многократном решении СЛАУ итерационным методом BiCGStab в прямом (N^+_{it}) и обратном (N^-_{it}) порядке для структур: 1 – при N = 1600 (*а*), 3200 (*б*); 2 – при N = 2001 (*в*), 3001 (*г*); 3 – при N = 1709 (*д*), 3109 (*е*)

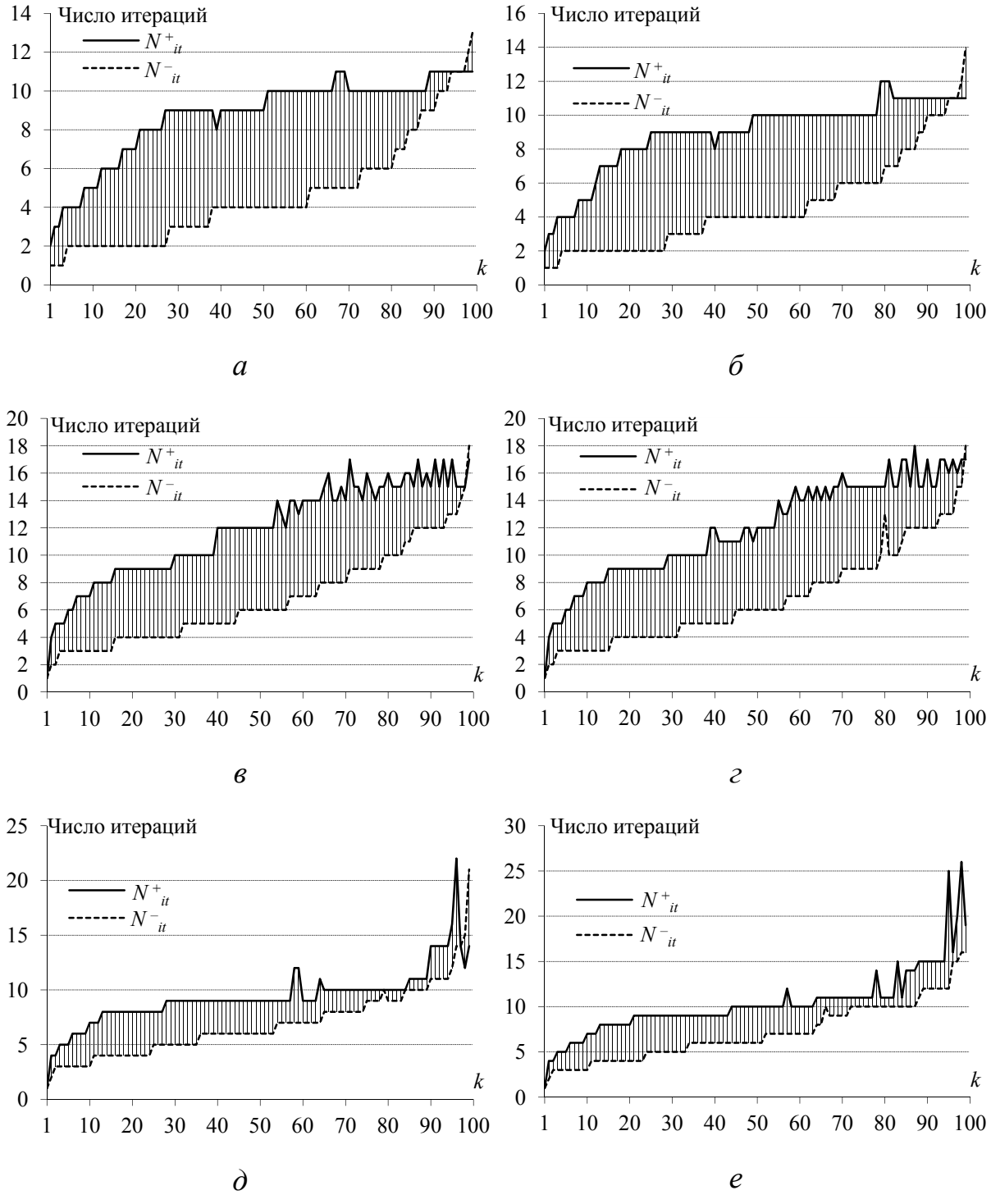


Рисунок 3.24 – Число итераций при многократном решении СЛАУ итерационным методом CGS в прямом (N^+_{it}) и обратном (N^-_{it}) порядке для структур: 1 – при N = 1600 (*а*), 3200 (*б*); 2 – при N = 2001 (*в*), 3001 (*г*); 3 – при N = 1709 (*д*), 3109 (*е*)

3.4. Использование выбора предобусловливателя

3.4.1. Алгоритм с выбором предобусловливателя

Одним из факторов, влияющим на многократное решение СЛАУ, является выбор предобусловливателя [135, 138]. Очевидно, что он будет влиять на общее время многократного решения СЛАУ, так как предобусловливатель будет меняться.

Как правило, на практике значения параметра структуры меняют в заданном диапазоне, что позволяет выбирать значение параметра для формирования предобусловливателя. Самый простой способ — выбрать среднее значение параметра для формирования предобусловливателя. На основе предложенного подхода разработан алгоритм 3.6. В нем в качестве способа формирования матрицы предобусловливания использовано LU-разложение.

Алгоритм 3.6 – Многократное решение СЛАУ с выбором матрицы для формирования предобусловливателя

1 Выбрать матрицу (i) для вычисления предобусловливателя, например $i = m / 2$
2 Вычислить матрицу предобусловливания $\mathbf{M}$ из матрицы $\mathbf{A}_i$
3 Для k от 1 до m
4 Найти $\mathbf{x}_k$ из уравнения $\mathbf{MA}_k\mathbf{x}_k = \mathbf{Mb}$ с заданной точностью Tol
5 Увеличить k

3.4.2. Вычислительный эксперимент

Использовался персональный компьютер, параметры которого указаны в п. 3.2.2, и итерационный метод BiCGStab. Рассматривались структура 1 (см. рисунок 3.4), структура 2 (см. рисунок 3.12) и структура 3 (см. рисунок 3.15). Характеристики матриц приведены в таблицах 3.6 и 3.9.

Сначала проведен вычислительный эксперимент с выбором 50-й матрицы для предобусловливателя и порядка решения с 1-й по 100-ю СЛАУ.

В таблице 3.12 представлены полученные ускорения относительно алгоритма с выбором 1-й матрицы для предобусловливания.

Таблица 3.12 – Ускорение решения 100 СЛАУ методам BiCGStab с использованием выбора 50-й матрицы для предобусловливателя

Структура	N	BiCGStab
1	1600	2,07
	3200	2,21
2	2001	2,14
	3001	1,94
3	1709	1,81
	3109	1,86

Видно, что для всех структур и N наблюдается ускорение (до 2,21 для структуры 2 при $N = 3200$). Оно достигается снижением общего числа итераций при многократном решении СЛАУ. На рисунке 3.25 приведено число итераций при решении k-й СЛАУ для случаев с предобусловливателем, полученным из первой ($\mathbf{M}^1$) и 50-й ($\mathbf{M}^{50}$) матриц СЛАУ.

Из рисунка следует, что число итераций при формировании предобусловливателя из 50-й матрицы СЛАУ значительно меньше, чем при формировании из 1-й, эта разница и объясняет полученное ускорение.

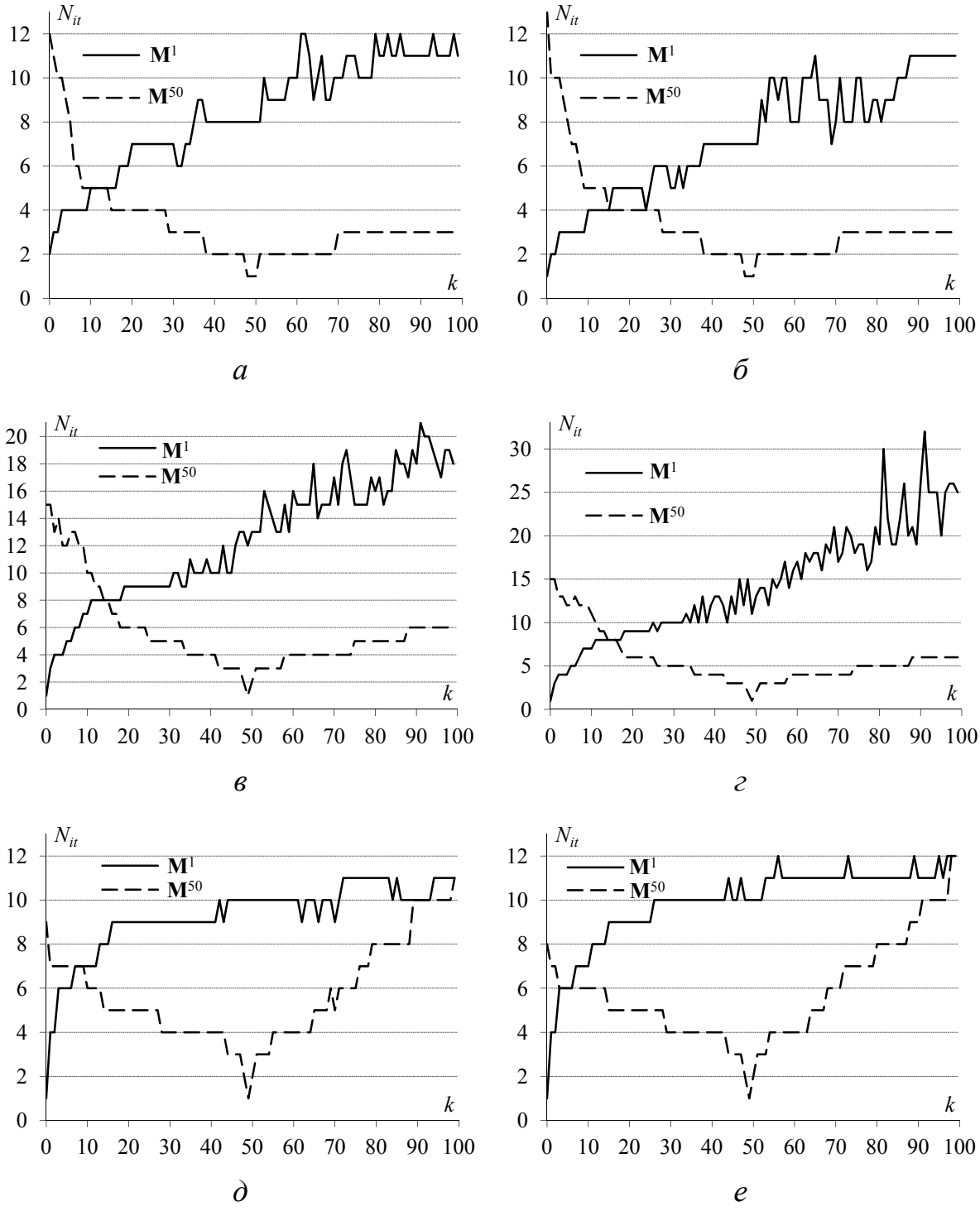


Рисунок 3.25 – Число требуемых итераций в зависимости от k при использовании предобусловливателя, полученного из первой ($\mathbf{M}^1$) и 50-й ($\mathbf{M}^{50}$) матриц СЛАУ для структур:
1 – при $N = 1600$ (*а*), 3200 (*б*);
2 – при $N = 2001$ (*в*), 3001 (*г*);
3 – при $N = 1709$ (*д*), 3109 (*е*)

4. Комплекс программ для решения СЛАУ итерационными методами

4.1. Расчет сложности алгоритмов

Расчет оценки сложности алгоритмов выполнен с использованием данных таблицы 3.8.

4.1.1. LU-разложение

Для расчета сложности использовался алгоритм LU-разложения *ikj*-версии.

Оценка по О-нотации

$$f_{LU}(N) = \sum_{i=2}^{N}\sum_{k=1}^{i-1}\sum_{j=k+1}^{N}1 = \frac{1}{6}N^3.$$

Оценка по арифметической сложности

Расчет числа элементарных операций алгоритма выполнен по их видам.

Деление:

$$f(N) = \sum_{i=2}^{N}\left(\sum_{k=1}^{i-1}(1)\right) = \sum_{i=1}^{N-1}(i-1+1) = \frac{N^2-N}{2}.$$

Умножение и сложение (вычитание) (в алгоритме их количество одинаково):

$$f(N)=\sum_{i=2}^{N}\left(\sum_{k=1}^{i-1}\left(\sum_{j=k+1}^{N}(1)\right)\right)=\sum_{i=2}^{N}\left(\sum_{k=1}^{i-1}\left(\sum_{j=1}^{N}(1)-\sum_{j=1}^{k}(1)\right)\right)=\sum_{i=2}^{N}\left(\sum_{k=1}^{i-1}(N-k)\right)=$$

$$=\sum_{i=2}^{N}\left(\sum_{k=1}^{i-1}N-\sum_{k=1}^{i-1}k\right)=\sum_{i=2}^{N}\left(N(i-1)-\frac{(i-1)i}{2}\right)=\sum_{i=2}^{N}\left(Ni-N-\frac{i^2-i}{2}\right)=$$

$$=\sum_{i=2}^{N}(Ni)-\sum_{i=2}^{N}(N)-\frac{1}{2}\sum_{i=2}^{N}(i^2)+\frac{1}{2}\sum_{i=2}^{N}(i)=N\sum_{i=1}^{N-1}(i+1)-N(N-1)-$$

$$-\frac{1}{2}\sum_{i=1}^{N-1}(i+1)^2+\frac{1}{2}\sum_{i=1}^{N-1}(i+1)=N\left(\sum_{i=1}^{N-1}(i)+N-1\right)-N^2+N-$$

$$-\frac{1}{2}\left(\sum_{i=1}^{N-1}\left(i^2+2i+1\right)\right)+\frac{1}{2}\left(\sum_{i=1}^{N-1}(i)+N-1\right)=N\left(\sum_{i=1}^{N-1}(i)+N-1\right)-$$

$$-N^2+N-\frac{1}{2}\left(\sum_{i=1}^{N-1}i^2+2\sum_{i=1}^{N-1}i+(N-1)\right)+\frac{1}{2}\left(\sum_{i=1}^{N-1}(i)+N-1\right)=$$

$$=N\left(\frac{N(N-1)}{2}+N-1\right)-N^2+N-\frac{1}{2}\left(\frac{(N-1)N(2(N-1)+1)}{6}+2\frac{N(N-1)}{2}+N-1\right)+$$

$$+\frac{1}{2}\left(\frac{N(N-1)}{2}+N-1\right)=N\frac{N^2-N+2N-2}{2}-N^2+N-$$

$$-\frac{1}{2}\left(\frac{(N^2-N)(2N-1)+6N^2-6N+6N-6}{6}\right)+\frac{N^2-N+2N-2}{4}=$$

$$=\frac{N^3+N^2-2N}{2}-N^2+N-\frac{1}{2}\left(\frac{2N^3-N^2-2N^2+N+6N^2-6}{6}\right)+$$

$$+\frac{N^2+N-2}{4}=\frac{N^3+N^2-2N-2N^2+2N}{2}-$$

$$-\left(\frac{2N^3-N^2-2N^2+N+6N^2-6}{12}\right)+\frac{3N^2+3N-6}{12}=$$

$$=\frac{N^3-N^2}{2}-\left(\frac{2N^3+3N^2+N-6}{12}\right)+\frac{3N^2+3N-6}{12}=$$

$$=\left(\frac{6N^3-6N^2-2N^3-3N^2-N+6+3N^2+3N-6}{12}\right)=$$

$$=\left(\frac{6N^3-2N^3-6N^2-3N^2+3N^2-N+3N-6+6}{12}\right)=$$

$$=\frac{4N^3-6N^2+2N}{12}=\frac{2N^3-3N^2+N}{6}.$$

Инкремент и сравнение (в алгоритме их количество одинаково):

$$f(N) = (N-1)+\sum_{i=2}^{N}\left((i-1)+\sum_{k=1}^{i-1}(N-k)\right)=(N-1)+\sum_{i=2}^{N}\left((i-1)+\sum_{k=1}^{i-1}(N)-\sum_{k=1}^{i-1}(k)\right)=$$

$$=(N-1)+\sum_{i=2}^{N}\left((i-1)+N(i-1)-\frac{(i-1)i}{2}\right)=(N-1)+\sum_{i=2}^{N}\left(\frac{2i-2+2Ni-2N-i^2+i}{2}\right)=$$

$$=(N-1)+\sum_{i=2}^{N}\left(\frac{2Ni-2N-i^2+3i-2}{2}\right)=$$

$$=(N-1)+\frac{1}{2}\left(\sum_{i=2}^{N}(2Ni)-\sum_{i=2}^{N}(2N)-\sum_{i=2}^{N}\left(i^2\right)+3\sum_{i=2}^{N}(i)-\sum_{i=2}^{N}(2)\right)=$$

$$=(N-1)+\frac{1}{2}\left(2N\sum_{i=1}^{N-1}(i+1)-2N\sum_{i=1}^{N-1}(1)-\sum_{i=1}^{N-1}\left((i+1)^2\right)+3\sum_{i=1}^{N-1}(i+1)-2\sum_{i=1}^{N-1}(1)\right)=$$

$$=(N-1)+\frac{1}{2}\left(2N\left(\frac{N-1}{2}N+N-1\right)-2N(N-1)-\sum_{i=1}^{N-1}\left(i^2+2i+1\right)+\right.$$

$$\left.+3\frac{N-1}{2}N+3N-3-2N+2\right)=(N-1)+\frac{1}{2}\left(2N\left(\frac{N^2-N+2N-2}{2}\right)-2N^2+2N-\right.$$

$$\left.-\left(\frac{(N-1)N(2N-1)}{6}+2\frac{N-1}{2}N+N-1\right)+\frac{3N^2-3N+2N-2}{2}\right)=$$

$$=(N-1)+\frac{1}{2}\left(\left(\frac{2N^3+2N^2-4N}{2}\right)-2N^2+2N-\right.$$

$$\left.-\left(\frac{(N^2-N)(2N-1)}{6}+\frac{2N^2-2N+2N-2}{2}\right)+\frac{3N^2-N-2}{2}\right)=$$

$$=(N-1)+\frac{1}{2}\left(\frac{2N^3+2N^2-4N-4N^2+4N}{2}-\right.$$

$$-\left(\frac{2N^3-N^2-2N^2+N}{6}+\frac{6N^2-6N+6N-6}{6}\right)+\frac{9N^2-3N-6}{6}\Bigg)=$$

$$=(N-1)+\frac{1}{2}\left(\left(\frac{2N^3-2N^2}{2}\right)-\left(\frac{2N^3-3N^2+N}{6}+\frac{6N^2-6}{6}\right)+\frac{9N^2-3N-6}{6}\right)=$$

$$=(N-1)+\frac{1}{2}\left(\left(\frac{6N^3-6N^2}{6}\right)-\left(\frac{2N^3+3N^2+N-6}{6}\right)+\frac{9N^2-3N-6}{6}\right)=$$

$$=(N-1)+\frac{1}{2}\left(\frac{6N^3-6N^2-2N^3-3N^2-N+6+9N^2-3N-6}{6}\right)=$$

$$=(N-1)+\left(\frac{4N^3-4N}{12}\right)=\frac{4N^3-4N+12N-12}{12}=\frac{4N^3+8N-12}{12}=$$

$$=\frac{N^3+2N-3}{3}.$$

Присваивание:

$$f(N)=1+\sum_{i=2}^{n}\left(1+\sum_{k=1}^{i-1}(1+n-k)\right)=1+\sum_{i=2}^{N}\left(1+i-1+Ni-N-\frac{(i-1)i}{2}\right)=$$

$$=1+\sum_{i=2}^{N}\left(i+Ni-N-\frac{i^2-i}{2}\right)=1+\frac{1}{2}\sum_{i=2}^{N}\left(3i+2Ni-2N-i^2\right)=$$

$$=1+\frac{1}{2}\left(3\sum_{i=2}^{N}i+2N\sum_{i=2}^{N}i-2N\sum_{i=2}^{N}1-\sum_{i=2}^{N}i^2\right)=$$

$$=1+\frac{1}{2}\left(3\sum_{i=1}^{N-1}(i+1)+2N\sum_{i=1}^{N-1}(i+1)-2N(N-1)-\sum_{i=1}^{N-1}(i+1)^2\right)=$$

$$=1+\frac{1}{2}\left(3\frac{(N-1)N}{2}+3N-3+2N\frac{(N-1)N}{2}+2N^2-2N-2N(N-1)-\right.$$

$$\left.-\sum_{i=1}^{N-1}\left(i^2+2i+1\right)\right)=1+\frac{1}{2}\left(3\frac{N^2-N}{2}+3N-3+2N\frac{N^2-N}{2}+2N^2-2N-2N^2+\right.$$

$$\left.+2N-\sum_{i=1}^{N-1}i^2-2\sum_{i=1}^{N-1}i-\sum_{i=1}^{N-1}1\right)=1+\frac{1}{2}\left(\frac{3N^2-3N}{2}+3N-3+\frac{2N^3-2N^2}{2}-\right.$$

$$-\frac{(N-1)N(2N-1)}{6}-2\frac{(N-1)N}{2}-N+1\Bigg)=$$

$$=1+\frac{1}{2}\left(\frac{2N^3+N^2+3N-6}{2}-\frac{2N^3-3N^2+N}{6}-\frac{2N^2-2N-2N+2}{2}\right)=$$

$$=1+\frac{1}{2}\left(\frac{2N^3-N^2+7N-8}{2}-\frac{2N^3-3N^2+N}{6}\right)=$$

$$=1+\frac{1}{2}\left(\frac{6N^3-3N^2+21N-24}{6}-\frac{2N^3-3N^2+N}{6}\right)=$$

$$-1+\frac{1}{2}\left(\frac{4N^3+20N-24}{6}\right)=\frac{4N^3+20N-12}{12}=\frac{N^3+5N-3}{3}.$$

В таблице 4.1 приведено полученное количество элементарных операций алгоритма.

Таблица 4.1 – Количество операций алгоритма LU-разложения

Операция	Количество арифметических операций
Деление	$\frac{N^2-N}{2}$
Умножение	$\frac{2N^3-3N^2+N}{6}$
Инкремент	$\frac{N^3+2N-3}{3}$
Сложение (вычитание)	$\frac{2N^3-3N^2+N}{6}$
Сравнение	$\frac{N^3+2N-3}{3}$
Операции присваивания	$\frac{N^3+5N-3}{3}$
Итого	$\left(10N^3-3N^2+16N-18\right)/6$

4.1.2. BiCGStab

При расчете сложности алгоритма BiCGStab использовался алгоритм 1.2. Учитывалось наличие двух возможных условий выхода из итерационного процесса (строки 17 и 24).

Оценка по *O*-нотации

BiCGStab, условие 1:

$$f(N, N_{it}) = 4N^2 + 6N + 4(N_{it} - 1)\left(N^2 + 2N\right).$$

BiCGStab, условие 2:

$$f(N, N_{it}) = 2N^2 + 2N + 4N_{it}\left(N^2 + 2N\right).$$

Оценка по арифметической сложности

В таблице 4.2 приведено полученное количество элементарных операций алгоритма.

Таблица 4.2 – Количество операций алгоритма BiCGStab

Операция	Условие 1	Условие 2
Деление	$N_{it} \cdot (4 + N) + (N_{it} - 1) \cdot (N + 2)$	$N_{it} \cdot (2N + 6)$
Умножение	$N^2 + N_{it} \cdot (N + 1 + 2N + N^2 - N + N^2 + N + 2N) + N + (N_{it} - 1) \cdot (N^2 - N + N^2 + 6N) = N^2 + N + N_{it} \cdot (2N^2 + 5N + 1) + (N_{it} - 1) \cdot (2N^2 + 5N)$	$N^2 + N_{it} \cdot (N + 1 + 2N + N^2 - N + N^2 + N + 2N + N^2 - N + N^2 + 6N) = N^2 + N_{it} \cdot (4N^2 + 10N + 1)$
Инкремент	$N^2 + N + N_{it} + N_{it} \cdot (2N + N^2 + N - 1 + N^2 + 2N) + N + (N_{it} - 1) \cdot (N^2 + N - 1 + N + N^2 + 4N) = N^2 + 2N + N_{it} \cdot (2N^2 + 5N) + (N_{it} - 1) \times (2N^2 + 6N - 1)$	$N^2 + N + N_{it} + N_{it} \cdot (2N + N^2 + N - 1 + N^2 + 2N + N^2 + N - 1 + N + N^2 + 4N) = N^2 + N + N_{it} \cdot (4N^2 + 11N - 1)$
Сложение (вычитание)	$N^2 + N + N_{it} \cdot (N + 2N + N^2 - N + N^2 + N + 2N) + N + (N_{it} - 1) \times (N^2 - N + N^2 + 2N + 4N) = N^2 + 2N + N_{it} \cdot (2N^2 + 5N) + (N_{it} - 1) \cdot (2N^2 + 5N)$	$N^2 + N + N_{it} \cdot (N + 2N + N^2 - N + N^2 + N + 2N + N^2 - N + N^2 + 2N + 4N) = N^2 + N + N_{it} \cdot (4N^2 + 10N)$

Окончание таблицы 4.2

Операция	Условие 1	Условие 2
Сравнение	$N^2 + N + N_{it} + N_{it} \cdot (2N^2 + 5N - 1 + 1) + N + (N_{it} - 1) \cdot (2N^2 + 2N - 1 + 4N + 1) = N^2 + 2N + N_{it} \cdot (2N^2 + 5N + 1) + (N_{it} - 1) \cdot (2N^2 + 6N)$	$N^2 + N + N_{it} + N_{it} \cdot (2N^2 + 5N - 1 + 1 + 2N^2 + 2N - 1 + 4N + 1) = N^2 + N + N_{it} \cdot (4N^2 + 11N + 1)$
Операции присваивания	$N^2 + 5N + 2 + N_{it} \cdot (N + 2 + 2 + N + 2N^2 + 3N + 1 + 3N + 2 + 2N + 3) + 1 + N + (N_{it} - 1) \cdot (N^2 + 3N + 1 + 1 + 2N + N^2 + 3 + 2N + 1 + N + N + 1 + 2 + N + 1) = N^2 + 6N + 3 + N_{it} \cdot (2N^2 + 10N + 10) + (N_{it} - 1) \times \times (2N^2 + 10N + 10)$	$N^2 + 5N + 2 + N_{it} \cdot (N + 2 + 2 + N + 2N^2 + 3N + 1 + 3N + 2 + 2N + 3 + N^2 + 3N + 1 + 1 + 2N + N^2 + 3 + 2N + 1 + N + N + 1 + 2 + N + 1) = N^2 + 5N + 2 + N_{it} \cdot (4N^2 + 20N + 20)$
Итого	$5N^2 + 13N + 3 + N_{it} (10N^2 + 31N + 16) + (N_{it} - 1) \cdot (10N^2 + 33N + 11)$	$5N^2 + 8N + 2 + N_{it} \cdot (20N^2 + 64N + 37)$

4.1.3. CGS

При расчете сложности алгоритма CGS использовался алгоритм 1.3.

Оценка по *O*-нотации

$$f(N, N_{it}) = 2N^2 + N_{it}\left(8N^2 + 5N\right).$$

Оценка по арифметической сложности

В таблице 4.3 приведено полученное количество элементарных операций алгоритма.

Таблица 4.3 – Количество операций алгоритма CGS

Операция	Количество арифметических операций
Деление	$N_{it} \cdot (2N + 2) + N_{it} - 1$
Умножение	$2N^2 + N + N_{it} \cdot (N + N^2 - N + N^2 + N + N + N^2 - N + + 2N + N^2 + N) + (N_{it} - 1) \cdot (3N) = 2N^2 + N + N_{it} \cdot (4N^2 + + 3N) + (N_{it} - 1) \cdot (3N)$
Инкремент	$2N^2 + N + N_{it} \cdot (1+N+N+ N^2 + N - 1 + N + N^2 + N^2 + N - - 1+3N+N^2+ N) + (N_{it}-1)\cdot(N) = = 2N^2 + N + N_{it} \cdot (4 N^2+9 N -1) + (N_{it}-1)\cdot(N)$
Сложение (вычитание)	$2N^2 + 3N + N_{it} \cdot (N + N^2 - N + N + N^2 + N + N^2 - N + + 2N + N^2 + N) + (N_{it} - 1) \cdot (3N) = 2N^2 + 3N + + N_{it} \cdot (4N + 4N^2) + (N_{it} - 1) \cdot (3N)$
Сравнение	$2N^2 + N + N_{it} \cdot (1 + N + N^2 + N - 1 + N^2 + N + N + N^2 + + N - 1 + 3N + N^2 + 1 + N) + (N_{it} - 1) \cdot (1 + N) = = 2N^2 + N + N_{it} \cdot (9N + 4N^2) + (N_{it} - 1) \cdot (1 + N)$
Операции присваивания	$2N^2 + 10N + 1 + N_{it} \cdot (N + 2 + N^2 + 3N + 1 + 2 + 3N + + N^2 + 2 + N + N^2 + 3N + 1 + 3 + 4N + N^2 + 3 + N) + + (N_{it} - 1) \cdot (2 + 2N) = 2N^2 + 10N + 1 + + N_{it} \cdot (4N^2 + 16N + 14) + (N_{it} - 1) \cdot (2N + 2)$
Итого	$10N^2 + 16N + 1 + N_{it} \cdot (20N^2 + 43N + 15) + + (N_{it} - 1) \cdot (10N + 4)$

4.2. Программная реализация алгоритмов

Ниже приведены исходные коды программ, разработанных на основе алгоритмов, предложенных в разделах 2 и 3. Алгоритмы реализованы в среде разработки Microsoft Visual C++ 2008.

4.2.1. Форматы хранения разреженных матриц

ILU(0)-разложение матрицы, хранимой в формате CSR

Исходный код программы, реализующей алгоритм 2.2:

```
void LU_Alg(int n ,double* &aelem_MatrM,int* &jptr_MatrM,int* &iptr_MatrM,int &N_CSR)
{
  int k,i,j,s;
  int NachStroki, KonSroki, NachStroki1,KonSroki1, TekEl1;
```

```
double diaronEl;
bool Prodolgat,NashliElem;
for (i = 1;i < n;i++)
{
        for (k = 0;k <= (i–1);k++)
        {
                TekEl1 = –1;
                for (j = iptr_MatrM[k]; j <= iptr_MatrM[k+1];j++)
                {
                        if (jptr_MatrM[k] == k)
                        {
                                TekEl1 = j;
                                break;
                        }
                        if (jptr_MatrM[k] > k)
                                break;
                }
                diaronEl = aelem_MatrM[TekEl1];
                NashliElem = false;
                NachStroki = iptr_MatrM[i];
                KonSroki = iptr_MatrM[i+1];
                NachStroki1 = iptr_MatrM[k];
                KonSroki1 = iptr_MatrM[k+1];
                for (s = NachStroki;s < KonSroki;s++)
                {
                        if(jptr_MatrM[s] == k)
                        {
                                aelem_MatrM[s] = aelem_MatrM[s]/diaronEl;
                                TekEl1 = s;
                                NashliElem = true;
                                NachStroki = s;
                                break;
                        }
                        if (jptr_MatrM[s]>k)
                                break;
                }
                if (NashliElem)
                {
                        while (jptr_MatrM[NachStroki]<(k+1))
                                NachStroki++;
                        while (jptr_MatrM[NachStroki1]<(k+1))
                                NachStroki1++;
                        if (KonSroki1 <= NachStroki1)
```

```
                        continue;
                    if (KonSroki <= NachStroki)
                        continue;
                    Prodolgat = true;
                    while (Prodolgat)
                    {
                        if (jptr_MatrM[NachStroki] ==
                            jptr_MatrM[NachStroki1])
                        {
                            aelem_MatrM[NachStroki] –=
    aelem_MatrM[TekEl1] *   aelem_MatrM[NachStroki1];
                            NachStroki++;
                            NachStroki1++;
                        }
                        if
(jptr_MatrM[NachStroki]>jptr_MatrM[NachStroki1])
                            NachStroki1++;
                        if
(jptr_MatrM[NachStroki1]>jptr_MatrM[NachStroki])
                            NachStroki++;
                        if (NachStroki1 >= KonSroki1)
                            Prodolgat = false;
                        if (NachStroki >= KonSroki)
                            Prodolgat = false;
                    }
                }
            }
        }
}
```

ILU(0)-разложение матрицы с использованием формата CSR с дополнительным вектором Diag

Исходный код программы, реализующей алгоритм 2.4:

```
void LU_Alg_Diag(int n ,double* &aelem_MatrM,int* &jptr_MatrM,int*
&iptr_MatrM,int* &UdialM,int &N_CSR)
{
    int k,i, s;
    int NachStroki,KonSroki, NachStroki1,KonSroki1, TekEl1;
    double diaronEl;
    bool Prodolgat,NashliElem;
    for (i = 1;i<n;i++)
```

```
{
        for (k = 0;k <= (i–1);k++)
        {
                TekEl1 = UdialM[k];
                diaronEl = aelem_MatrM[TekEl1];
                NashliElem = false;
                NachStroki = iptr_MatrM[i];
                KonSroki = iptr_MatrM[i+1];
                NachStroki1 = iptr_MatrM[k];
                KonSroki1 = iptr_MatrM[k+1];
                for (s = NachStroki;s<KonSroki;s++)
                {
                        if(jptr_MatrM[s] == k)
                        {
                                aelem_MatrM[s] = aelem_MatrM[s]/diaronEl;
                                TekEl1 = s;
                                NashliElem = true;
                                NachStroki = s;
                                break;
                        }
                        if (jptr_MatrM[s] > k)
                                break;
                }
                if (NashliElem)
                {
                        while (jptr_MatrM[NachStroki] < (k+1))
                                NachStroki++;
                        while (jptr_MatrM[NachStroki1] < (k+1))
                                NachStroki1++;
                        if (KonSroki1 <= NachStroki1)
                                continue;
                        if (KonSroki <= NachStroki)
                                continue;
                        Prodolgat = true;
                        while (Prodolgat)
                        {
                                if (jptr_MatrM[NachStroki] ==
                                        jptr_MatrM[NachStroki1])
                                {
                                        aelem_MatrM[NachStroki] –=
aelem_MatrM[TekEl1] * aelem_MatrM[NachStroki1];
                                        NachStroki++;
                                        NachStroki1++;
```

```
                                        }
                                        if
(jptr_MatrM[NachStroki]>jptr_MatrM[NachStroki1])
                                                NachStroki1++;
                                        if
(jptr_MatrM[NachStroki1]>jptr_MatrM[NachStroki])
                                                NachStroki++;
                                        if (NachStroki1 >= KonSroki1)
                                                Prodolgat = false;
                                        if (NachStroki >= KonSroki)
                                                Prodolgat = false;
                                }
                        }
                }
        }
}
```

ILU(0)-разложение с использованием формата CSR с двумя дополнительными векторами

Исходный код программы, реализующей алгоритм 2.6:

```
void LU_Alg_Diag_Modif(int n ,double* &aelem_MatrM,int*
&jptr_MatrM,int* &iptr_MatrM,int* &UdialM,int &N_CSR)
{
        int k,i,j;
        int NachStroki, KonSroki, NachStrokiOsn, NachStroki1, KonSroki1,
TekEl1;
        bool ProdolgatOsn;
        bool *Line;
        int *Line1;
        Line = new bool [n];
        Line1 = new int [n];
        for (j = 0;j<n;j++)
        {
                Line[j] = false;
                Line1[j] = 0;
        }
        for (i = 1;i<n;i++)
        {
                NachStrokiOsn = iptr_MatrM[i];
                for (j = NachStrokiOsn;j < iptr_MatrM[i+1];j++){
                        Line[jptr_MatrM[j]] = true;
```

```
                Line1[jptr_MatrM[j]] = j;
            }
            ProdolgatOsn = true;
            while (ProdolgatOsn)
            {
                k = jptr_MatrM[NachStrokiOsn];
                if (k >= i)
                    break;
                TekEl1 = UdialM[k];
                aelem_MatrM[NachStrokiOsn] =
        aelem_MatrM[NachStrokiOsn] /aelem_MatrM[TekEl1];
                TekEl1 = NachStrokiOsn;
                NachStrokiOsn++;
                NachStroki = NachStrokiOsn;
                KonSroki = iptr_MatrM[i+1];
                NachStroki1 = UdialM[k];
                KonSroki1 = iptr_MatrM[k+1];
                NachStroki1++;
                if (KonSroki1 <= NachStroki1 || KonSroki <= NachStroki)
                    continue;
                for (j = NachStroki1;j < KonSroki1;j++)
                {
                    if (Line[jptr_MatrM[j]])
                        aelem_MatrM[Line1[jptr_MatrM[j]]] –=
                        aelem_MatrM[TekEl1]*aelem_MatrM[j];
                }
            }
            for (j = 0;j<n;j++)
                Line[j] = false;
        }
}
```

Решение СЛАУ с матрицей в модифицированном формате CSR

Исходный код программы, реализующей алгоритм 2.8:

```
void Solve_Mb_x(int n, double* &VecB,double* &VecX,double*
&aelem_MatrM,int* &UdialM, int* &jptr_MatrM, int* &iptr_MatrM,int
&N_CSR)
{
        double res1,res2,res3;
        int i,j,k,NachStroki,KonSroki;
```

```
//Прямой ход
double* VecXv;
VecXv = new double[n];
for (i = 0;i < n;i++)
        VecXv[i] = VecX[i];
for (i = 0;i < n;i++)
{
        NachStroki = iptr_MatrM[i];
        KonSroki = iptr_MatrM[i+1];
        VecB[i] = VecXv[i];
        for (j = NachStroki;j<KonSroki;j++)
        {
                k = jptr_MatrM[j];
                if (k >= i)
                        continue;
                if (i != 0)
                {
                        res1 = VecB[i];
                        res2 = VecXv[k];
                        res3 = aelem_MatrM[j];
                        VecB[i] = res1–res2*res3;
                }
        }
        VecXv[i] = VecB[i];
}
//Обратный ход
for (i = (n–1);i >= 0;i– –)
{
        NachStroki = iptr_MatrM[i];
        KonSroki = iptr_MatrM[i+1];
        for (j = NachStroki;j<KonSroki;j++)
        {
                k = jptr_MatrM[j];
                if (k <= i)
                        continue;
                VecB[i] = VecB[i]–VecXv[k]*aelem_MatrM[j];
        }
        res1 = aelem_MatrM[UdialM[i]];
        VecXv[i] = VecB[i]/res1;
}
for (i = 0;i<n;i++)
        VecB[i] = VecXv[i];
```

```
        delete [] VecXv;
}
```

4.2.2. Итерационные методы

BiCGStab

Исходный код программы, реализующей алгоритм 2.6:

```
for (i = 0; i < n; i++)
{
        r[i] = 0;
        for (j = 0; j < n; j++)
        {
                r[i] += A[i][j]*x0[j]
        }
        r[i] = b[i] – r[i]
        rtilda[i] = r[i];
        p[i] = r[i];
}
for (iter = 1; iter < Nitmax; iter++)
{
        ro = 0.0;
        for (uint i = 0; i < m; i++)
                ro += rtilda[i] * r[i];
        beta = (ro/ro_old)*(alpha/omega);
        for (uint i = 0; i < m; i++)
                p[i] = r[i] + beta * (p[i] – omega* v[i]);
        for (uint i = 0; i<m; i++)
        {
                ptilda[i] = p[i];
                for (uint j = 0; j < i; j++)
                        ptilda[i] –= ptilda[j]*ALU[i][j];
        }
        ptilda[m–1] /= ALU[m–1][m–1];
        for (int i = m–2; i >= 0; i– –)
        {
                for (uint j = i+1; j < m; j++)
                        ptilda[i] –= ALU[i][j]*ptilda[j];
                ptilda[i] /= ALU[i][i];
        }
        alpha = 0.0;
        for (uint i = 0; i < m; i++)
```

```
        {
                v[i] = 0.0;
                for (uint j = 0; j<m; j++)
                        v[i] += A[i][j] * ptilda[j];
                alpha += rtilda[i]*v[i];
        }
        alpha = ro/alpha;
        norm_ri = 0.0;
        for (uint i = 0; i < m; i++)
        {
                s[i] = r[i] – alpha * v[i];
                norm_ri += s[i]*s[i];
        }
        if (sqrt(norm_ri/norm_r0) <= Tol)
        {
                for (uint i = 0; i<m; i++)
                        x[i] += alpha*ptilda[i];
                break;
        }
        for (uint i = 0; i < m; i++)
        {
                t[i] = 0.0;
                for (uint j = 0; j<m; j++)
                        t[i] += A[i][j] * stilda[j];
        }
        tt = ts = 0.0;
        for (uint i = 0; i < m; i++)
        {
                ts += t[i]*s[i];
                tt += t[i]*t[i];
        }
        omega = ts/tt;
        for (uint i = 0; i < m; i++)
                x[i] +=  alpha * ptilda[i] + omega*stilda[i];
        for (uint i = 0; i < m; i++)
                r[i] = s[i] – omega * t[i];
        norm_ri = 0.0;
        for (uint i = 0; i < m; i++)
                norm_ri += r[i]*r[i];
        if (sqrt(norm_ri/norm_r0) <= Tol)
                break;
        ro_old = ro;
}
```

CGS

Исходный код программы, реализующей алгоритм 1.3:

```
for (int i = 0; i<m; i++)
{
        r[i] = 0.0;
        for (int j = 0; j < m; j++)
                r[i] += A[i][j] * 1.0;
        r[i] = b[i] – r[i];
        norm_r0 += r[i]*r[i];
        r[i] = 0.0;
        for (int j = 0; j < m; j++)
                r[i] += A[i][j] * x[j];
        r[i] = b[i] – r[i];
        rtilda[i] = r[i];
        p[i] = r[i];
        u[i] = r[i];
}
for (iter = 1; iter < Nitmax; iter++)
{
        ro = 0.0;
        for (int i = 0; i < m; i++)
                ro += rtilda[i] * r[i];
        if (iter > 1)
        {
                beta = (ro/ro_old);
                for (int i = 0; i < m; i++)
                {
                        u[i] = r[i] + beta * q[i];
                        p[i] = u[i] + beta * (q[i] + beta * p[i]);
                }
        }
        for (int i = 0; i < m; i++)
        {
                ptilda[i] = p[i];
                for (int j = 0; j<i; j++)
                        ptilda[i] –= ptilda[j] * ALU_Glob[i][j];
        }
        ptilda[m–1] /= ALU_Glob[m–1][m–1];
        for (int i = m–2; i >= 0; i– –)
        {
                for (int j = i+1; j < m; j++)
                        ptilda[i] –= ALU_Glob[i][j]*ptilda[j];
```

```
        ptilda[i] /= ALU_Glob[i][i];
}
alpha = 0.0;
for (int i = 0; i < m; i++)
{
        v[i] = 0.0;
        for (int j = 0; j < m; j++)
                v[i] += A[i][j] * ptilda[j];
        alpha += rtilda[i]*v[i];
}
alpha = ro/alpha;
for (int i = 0; i < m; i++)
{
        q[i] = u[i] – alpha * v[i];
}
for (int i = 0; i < m; i++)
{
        utilda[i] = u[i] + q[i];
        for (int j = 0; j < i; j++)
                utilda[i] –= utilda[j]*ALU_Glob[i][j];
}
utilda[m–1] /= ALU_Glob[m–1][m–1];
for (int i = m–2; i >= 0; i– –)
{
        for (int j = i+1; j < m; j++)
                utilda[i] –= ALU_Glob[i][j]*utilda[j];
        utilda[i] /= ALU_Glob[i][i];
}
for (int i = 0; i < m; i++)
        x[i] +=  alpha * utilda[i];
for (int i = 0; i < m; i++)
{
        qtilda[i] = 0.0;
        for (int j = 0; j < m; j++)
                qtilda[i] += A[i][j] * utilda[j];
}
for (int i = 0; i < m; i++)
        r[i] –= alpha * qtilda[i];
norm_ri = 0.0;
for (int i = 0; i < m; i++)
        norm_ri += r[i]*r[i];
if (sqrt(norm_ri/norm_r0) <= Tol)
{
```

```
            break;
        }
        ro_old = ro;
}
```

4.2.3. Многократное решение СЛАУ

Итерационный метод с предобусловливанием

Исходный код программы, реализующей алгоритм 3.1:

```
void LU_Alg(double ** &A,int &m)
{
        for (int i = 1; i < m; i++)
                for (int k = 0; k < i; k++)
                {
                        A[i][k] /= A[k][k];
                        for (int j = k+1; j < m; j++)
                                A[i][j] –= A[i][k]*A[k][j];
                }
}
void StandAlg(double ** &A, double** &ALU, int &n,double &tau,double
&Tol)
{
        for (i = 0; i < n; i++)
        {
                r[i] = 0;
                for (j = 0; j < n; j++)
                {
                        r[i] += A[i][j]*x0[j]
                }
                r[i] = b[i] – r[i]
                rtilda[i] = r[i];
                p[i] = r[i];
        }
        for (iter = 1; iter < Nitmax; iter++)
        {
                ro = 0.0;
                for (uint i = 0; i < m; i++)
                        ro += rtilda[i] * r[i];
                beta = (ro/ro_old)*(alpha/omega);
                for (uint i = 0; i < m; i++)
                        p[i] = r[i] + beta * (p[i] – omega* v[i]);
```

```
for (uint i = 0; i < m; i++)
{
        ptilda[i] = p[i];
        for (uint j = 0; j < i; j++)
                ptilda[i] –= ptilda[j]*ALU[i][j];
}
ptilda[m–1] /= ALU[m–1][m–1];
for (int i = m–2; i >= 0; i– –)
{
        for (uint j = i+1; j < m; j++)
                ptilda[i] –= ALU[i][j]*ptilda[j];
        ptilda[i] /= ALU[i][i];
}
alpha = 0.0;
for (uint i = 0; i < m; i++)
{
        v[i] = 0.0;
        for (uint j = 0; j < m; j++)
                v[i] += A[i][j] * ptilda[j];
        alpha += rtilda[i]*v[i];
}
alpha = ro/alpha;
norm_ri = 0.0;
for (uint i = 0; i < m; i++)
{
        s[i] = r[i] – alpha * v[i];
        norm_ri += s[i]*s[i];
}
if (sqrt(norm_ri/norm_r0) <= Tol)
{
        for (uint i = 0; i < m; i++)
                x[i] += alpha*ptilda[i];
        break;
}
for (uint i = 0; i < m; i++)
{
        t[i] = 0.0;
        for (uint j = 0; j < m; j++)
                t[i] += A[i][j] * stilda[j];
}
tt = ts = 0.0;
for (uint i = 0; i < m; i++)
{
```

```
                ts += t[i]*s[i];
                tt += t[i]*t[i];
            }
            omega = ts/tt;
            for (uint i = 0; i < m; i++)
                x[i] +=  alpha * ptilda[i] + omega*stilda[i];
            for (uint i = 0; i < m; i++)
                r[i] = s[i] – omega * t[i];
            norm_ri = 0.0;
            for (uint i = 0; i < m; i++)
                norm_ri += r[i]*r[i];
            if (sqrt(norm_ri/norm_r0) <= Tol)
                break;
            ro_old = ro;
            for (int i = 0;i < m;i++)
                delete [] A[i];
            delete [] A;
        }
}
Void LoadMatrix(int i, double** &A, int &n)
{
        string fn_A = "test";
        fn_A += toString(i);
        fn_A += ".txt";
        ifstream fs_matrixA(fn_A.c_str(), ios::in);
        fs_matrixA >> n;
        A = new double *[n];
        for (int i = 0; i < n; i++)
            A[i] = new double [n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                fs_matrixA >> A[i][j];
        fs_matrixA.close();
}
int main(void)
{
        int n;
        double Tol = 1.0e–8;
        double tau = 0;
        double **A;
        LoadMatrix(1, A, n);
        double **ALU;
        ALU = new double *[m];
```

```
        for (int i = 0; i < m; i++)
                ALU [i] = new double [m];
        for (int i = 0; i < m; i++)
                for (int j = 0; j < m; j++)
                        ALU [i][j] = A [i][j];
        LU_Alg(ALU, n);
        for (int i = 1;i <= 100;i++)
        {
                LoadMatrix(i, A, n);
                StandAlg(A, ALU, n, tau, Tol)
        }
        for (int i = 0;i < m;i++)
                delete [] ALU[i];
        delete [] ALU;
}
```

Переформирование матрицы предобусловливания по порогу числа итераций

Исходный код программы, реализующей алгоритм 3.2:

```
void LU_Alg(double ** &A,int &m)
{
        for (int i = 1; i < m; i++)
                for (int k = 0; k < i; k++)
                {
                        A[i][k] /= A[k][k];
                        for (int j = k+1; j < m; j++)
                                A[i][j] –= A[i][k]*A[k][j];
                }
}
void StandAlg(double ** &A, double** &ALU, int &n,double &tau,double
&Tol, int &NumbIter)
{
        for (i = 0; i < n; i++)
        {
                r[i] = 0;
                for (j = 0; j < n; j++)
                {
                        r[i] += A[i][j]*x0[j]
                }
                r[i] = b[i] – r[i]
                rtilda[i] = r[i];
```

```
        p[i] = r[i];
}
for (iter = 1; iter < Nitmax; iter++)
{
        NumbIter = iter;
        ro = 0.0;
        for (uint i = 0; i < m; i++)
                ro += rtilda[i] * r[i];
        beta = (ro/ro_old)*(alpha/omega);
        for (uint i = 0; i < m; i++)
                p[i] = r[i] + beta * (p[i] – omega* v[i]);
        for (uint i = 0; i < m; i++)
        {
                ptilda[i] = p[i];
                for (uint j = 0; j < i; j++)
                        ptilda[i] –= ptilda[j]*ALU[i][j];
        }
        ptilda[m–1] /= ALU[m–1][m–1];
        for (int i = m–2; i >= 0; i– –)
        {
                for (uint j = i+1; j < m; j++)
                        ptilda[i] –= ALU[i][j]*ptilda[j];
                ptilda[i] /= ALU[i][i];
        }
        alpha = 0.0;
        for (uint i = 0; i < m; i++)
        {
                v[i] = 0.0;
                for (uint j = 0; j < m; j++)
                        v[i] += A[i][j] * ptilda[j];
                alpha += rtilda[i]*v[i];
        }
        alpha = ro/alpha;
        norm_ri = 0.0;
        for (uint i = 0; i < m; i++)
        {
                s[i] = r[i] – alpha * v[i];
                norm_ri += s[i]*s[i];
        }
        if (sqrt(norm_ri/norm_r0) <= Tol)
        {
                for (uint i = 0; i < m; i++)
                        x[i] += alpha*ptilda[i];
```

```
                break;
            }
            for (uint i = 0; i < m; i++)
            {
                t[i] = 0.0;
                for (uint j = 0; j < m; j++)
                    t[i] += A[i][j] * stilda[j];
            }
            tt = ts = 0.0;
            for (uint i = 0; i < m; i++)
            {
                ts += t[i]*s[i];
                tt += t[i]*t[i];
            }
            omega = ts/tt;
            for (uint i = 0; i < m; i++)
                x[i] +=  alpha * ptilda[i] + omega*stilda[i];
            for (uint i = 0; i < m; i++)
                r[i] = s[i] – omega * t[i];
            norm_ri = 0.0;
            for (uint i = 0; i < m; i++)
                norm_ri += r[i]*r[i];
            if (sqrt(norm_ri/norm_r0) <= Tol)
                break;
            ro_old = ro;
            for (int i = 0;i < m;i++)
                delete [] A[i];
            delete [] A;
        }
}
Void LoadMatrix(int i, double** &A, int &n)
{
        string fn_A = "test";
        fn_A += toString(i);
        fn_A += ".txt";
        ifstream fs_matrixA(fn_A.c_str(), ios::in);
        fs_matrixA >> n;
        A = new double *[n];
        for (int i = 0; i < n; i++)
            A[i] = new double [n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                fs_matrixA >> A[i][j];
```

```
        fs_matrixA.close();
}
int main(void)
{
        int n;
        double Tol = 1.0e–8;
        double tau = 0;
        double **A;
        LoadMatrix(1, A, n);
        double **ALU;
        ALU = new double *[m];
        for (int i = 0; i < m; i++)
                ALU [i] = new double [m];
        for (int i = 0; i < m; i++)
                for (int j = 0; j < m; j++)
                        ALU [i][j] = A [i][j];
        LU_Alg(ALU, n);
        int MaxIt = 10;
        int &NumbIter = 1;
        for (int i = 1;i <= 100;i++)
        {
                if (MaxIt < NumbIter)
                {
                        for (int i = 0; i < m; i++)
                                for (int j = 0; j < m; j++)
                                        LU_Alg(ALU, n);
                        LU_Alg(ALU, n);
                }
                LoadMatrix(i, A, n);
                StandAlg(A, ALU, n, tau, Tol, NumbIter)
        }
        for (int i = 0;i < m;i++)
                delete [] ALU[i];
        delete [] ALU;
}
```

Переформирование матрицы предобусловливания по увеличению среднего арифметического времени решения

Исходный код программы, реализующей алгоритм 3.3:

```
int main(void)
{
```

```
int n;
double Tol = 1.0e–8;
double tau = 0;
double **A;
LoadMatrix(1, A, n);
double **ALU;
ALU = new double *[m];
for (int i = 0; i < m; i++)
        ALU [i] = new double [m];
for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
                ALU [i][j] = A [i][j];
DWORD begin, end, time01, SumTime = 0;
begin = GetTickCount();
LU_Alg(ALU, n);
StandAlg(A, ALU, n, tau, Tol, NumbIter)
end = GetTickCount();
SumTime = (end – begin);
int &NumbIter = 1;
for (int i = 2;i <= 100;i++)
{
        LoadMatrix(i, A, n);
        begin = GetTickCount();
        StandAlg(A, ALU, n, tau, Tol, NumbIter)
        end = GetTickCount();
        time01 = (end – begin);
        if ((SumTime + time01)/i > SumTime / (i–1))
        {
                for (int i = 0; i < m; i++)
                        for (int j = 0; j < m; j++)
                                LU_Alg(ALU, n);
                begin = GetTickCount();
                LU_Alg(ALU, n);
                end = GetTickCount();
                SumTime += time01;
                time01 = (end – begin);
                SumTime += time01;
        }
        else
                SumTime += time01;
}
for (int i = 0;i < m;i++)
```

```
        delete [] ALU[i];
    delete [] ALU;
```

Переформирование предобусловливателя при увеличении средней сложности решения

Исходный код программы, реализующей алгоритм 3.4:

```
int main(void)
{
    int n;
    double Tol = 1.0e–8;
    double tau = 0;
    double **A;
    LoadMatrix(1, A, n);
    double **ALU;
    int SumSl = 0;
    ALU = new double *[m];
    for (int i = 0; i < m; i++)
        ALU [i] = new double [m];
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            ALU [i][j] = A [i][j];
    LU_Alg(ALU, n);
    StandAlg(A, ALU, n, tau, Tol, NumbIter)
    SumSl += n * n * n /6;
    int TekSl = 0;
    int &NumbIter = 1;
    for (int i = 2;i <= 100;i++)
    {
        LoadMatrix(i, A, n);
        StandAlg(A, ALU, n, tau, Tol, NumbIter)
        TekSl = 2*n*n + 2*n + NumbIter *(4*n*n+8*n)
        if ((SumSl + TekSl)/i > SumSl / (i–1))
        {
            for (int i = 0; i < m; i++)
                for (int j = 0; j < m; j++)
                    LU_Alg(ALU, n);
            LU_Alg(ALU, n);
            SumSl += TekSl;
            SumSl += n * n * n /6;
        }
        else
```

```
                SumSl += TekSl;
        }
        for (int i = 0;i < m;i++)
                delete [] ALU[i];
        delete [] ALU;
```

Выбор очередности решения

Исходный код программы, реализующей алгоритм 3.5:

```
int main(void)
{
        int n;
        double Tol = 1.0e–8;
        double tau = 0;
        double **A;
        LoadMatrix(1, A, n);
        double **ALU;
        ALU = new double *[m];
        for (int i = 0; i < m; i++)
                ALU [i] = new double [m];
        for (int i = 0; i < m; i++)
                for (int j = 0; j < m; j++)
                        ALU [i][j] = A [i][j];
        int &NumbIter = 1;
        // Если выбран прямой порядок
        if (Pr == true)
                for (int i = 1;i <= 100;i++)
                {
                        LoadMatrix(i, A, n);
                        if (i == 1)
                                LU_Alg(ALU, n);
                        StandAlg(A, ALU, n, tau, Tol, NumbIter)
                }
        else
                for (int i = 100;i >= 1;i– –)
                {
                        LoadMatrix(i, A, n);
                        if (i == 100)
                                LU_Alg(ALU, n);
                        StandAlg(A, ALU, n, tau, Tol, NumbIter)
                }
        for (int i = 0;i < m;i++)
```

```
        delete [] ALU[i];
    delete [] ALU;
```

Выбор матрицы для формирования предобусловливателя

Исходный код программы, реализующей алгоритм 3.6:

```
int main(void)
{
    int n;
    double Tol = 1.0e–8;
    double tau = 0;
    double **A;
// Номер матрицы для формирования предобусловливателя
    int Form = 50;
    LoadMatrix(Form, A, n);
    double **ALU;
    ALU = new double *[m];
    for (int i = 0; i < m; i++)
        ALU [i] = new double [m];
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            ALU [i][j] = A [i][j];
    LU_Alg(ALU, n);
    int &NumbIter = 1;
    for (int i = 1;i <= 100;i++)
    {
        LoadMatrix(i, A, n);
        StandAlg(A, ALU, n, tau, Tol, NumbIter)
    }
    for (int i = 0;i < m;i++)
        delete [] ALU[i];
    delete [] ALU;
```

Заключение

Монография посвящена проблеме многократного решения СЛАУ итерационными методами. Авторы надеются, что их работа будет полезной для читателя.

Обзор проблем ЭМС показал, что большая часть времени имитационного моделирования приходится на решение СЛАУ, поэтому, уменьшая время решения СЛАУ, можно сократить время моделирования в целом. Анализ методов решения СЛАУ выявил перспективность использования итерационных методов подпространства Крылова с применением предобусловливателя. При предобусловливании применяют разреженную матрицу, полученную из исходной путем предфильтрации. Для хранения разреженной матрицы предложен формат CSR, тем самым была достигнута экономия машинной памяти и времени решения СЛАУ.

На практике часто требуется многократное решение СЛАУ, что очень затратно по времени. Из анализа литературных источников следует, что для решения таких задач можно использовать итерационные методы с предобусловливанием и в перспективе обеспечить снижение временных затрат. На основе предложенных способов ускорения разработан алгоритм многократного решения СЛАУ. Сформулированы условия переформирования предобусловливателя при многократном решении СЛАУ. Разработан алгоритм многократного решения СЛАУ с выбором очередности решения и матрицы для формирования предобусловливателя.

Литература

1. ГОСТ Р 50397–2011. Совместимость технических средств электромагнитная. Термины и определения. – М. : Стандартинформ, 2013. – 57 с.

2. Князев, А.Д. Конструирование радиоэлектронной и электронновычислительной аппаратуры с учетом электромагнитной совместимости / А.Д. Князев, Л.Н Кечиев, Б.В. Петров. – М.: Радио и связь, 1989. – 224 с.

3. Газизов, Т.Р. Электромагнитный терроризм на рубеже тысячелетий / под ред. Т.Р. Газизова. – Томск: Томский государственный университет, 2002. – 206 с.

4. Cendes, Z. Simulating the behavior of high-speed circuits / Z. Cendes // Computer Design. – 1995. – V.34 (8). – P. 130–131.

5. Газизов, Т.Р. Основы автоматизации проектирования радиоэлектронных устройств : учеб. пособие / Т.Р. Газизов. – Томск: Томск. межвузовский центр дистанц. образования, 2005. – 243 с.

6. Сосунов, Б.В. Применение метода конечных разностей временной области в задачах дифракции радиоволн / Б.В. Сосунов, А.А. Тимчук // Вопросы ЭМС и расчета антенн и радиолиний. – СПб.: ВАС, 1994. – С. 220–226.

7. Харрингтон, Р.Ф. Применение матричных методов к задачам теории поля / Р.Ф. Харрингтон // ТИИЭР. – 1967. – № 2. – С. 5–19.

8. Silvester, P. Finite element solution of saturate magnetic field problems / P. Silvester, M. Chari // IEEE Trans. Power Appar. Syst. – 1970. – V. 89, № 7. – P. 1642–1651.

9. Weiland, T. A Discretization Method for the Solution of Maxwell's Equations for Six-Component Fields / T. Weiland // Electronics and Communications AEUE. – 1977. – V. 31(3). – P. 116–120.

10. Johns, P.B. Numerical solution of 2-dimensional scattering problems using a transmission line matrix / P.B. Johns, R.L. Beurle // Proceedings of the IEEE. – 1971. – V. 118(9). – P. 1203–1208.

11. Агапов, С.В. Электронные САПР для моделирования электромагнитных излучений от межсоединений печатных плат / С.В. Агапов // Проблемы электромагнитной совместимости тех-

нических средств: сб. докл. Всерос. симпозиума. – М., 2002. – С. 11–13.

12. Григорьев, А.Д. Методы вычислительной электродинамики / А.Д. Григорьев. – М.: Физматлит, 2013. – 430 с.

13. Газизов, Т.Р. Уменьшение искажений электрических сигналов в межсоединениях и влияний преднамеренных силовых электромагнитных воздействий: дис. … д-ра техн. наук / Газизов Тальгат Рашитович. – Томск, 2010. – 351 с.

14. Газизов, Т.Р. Вычисление емкостной матрицы двумерной конфигурации проводников и диэлектриков с ортогональными границами / Т.Р. Газизов // Известия вузов. Физика. – 2004. – № 3. – С. 88–90.

15. Газизов, Т.Р. Матрица емкостных коэффициентов трехмерной системы проводников и диэлектриков / Т.Р. Газизов // Известия вузов. Физика. – 1998. – № 3. – С. 123–125.

16. Газизов, Т.Р. Уменьшение искажений электрических сигналов в межсоединениях / Т.Р. Газизов ; под ред. Н.Д. Малютина.– Томск: НТЛ, 2003. – 212 с.

17. Wideband frequency-domain characterization of FR-4 and time-domain causality / A.R. Djordjevich, R.M. Biljic, V.D. Likar-Smiljanic, T.K. Sarkar // IEEE Transactions on Electromagnetic Compatibility. – 2001. – Vol. 43, № 4. – P. 662–666.

18. Куксенко, С.П. Совершенствование алгоритма вычисления методом моментов емкостных матриц структуры проводников и диэлектриков в диапазоне значений диэлектрической проницаемости / С.П. Куксенко, Т.Р. Газизов // Электромагнитные волны и электронные системы. – 2012. – № 10. – С. 13–21.

19. Суровцев, Р.С. Исследование ускорения многократного решения СЛАУ с частично изменяющейся матрицей блочным методом / Р.С. Суровцев, В.К. Салов // Электромагнитные волны и электронные системы. – 2012. – Т. 17(10). – С. 22–24.

20. Суровцев, Р.С. Вычисление матрицы емкостей произвольной системы проводников и диэлектриков методом моментов: зависимость ускорения за счет блочного LU-разложения от порядка матрицы СЛАУ / Р.С. Суровцев, С.П. Куксенко // Известия высших учебных заведений. Физика. – 2012. – Т. 55 (9/3). – С. 126–130.

21. Новые возможности системы моделирования электромагнитной совместимости TALGAT / С.П. Куксенко, А.М. Заболоцкий, А.О. Мелкозеров, Т.Р. Газизов // Докл. Томск. гос. ун-та систем упр. и радиоэлектроники. – 2015. – № 2(36). – С. 45–50.

22. Gazizov, T.R. Analytic expressions for Mom calculation of capacitance matrix of two dimensional system of conductors and dielectrics having arbitrary oriented boundaries / T.R. Gazizov // Proc. of the 2001 IEEE EMC Symposium, Montreal, Canada, August 13–17, 2001. – 2001. – Vol. 1. – P. 151–155.

23. Salov, V.K. Convergence of multiple iterative solution of linear algebraic systems with a fully varying matrix using a single calculated initial preconditioner / V.K. Salov, T.R. Gazizov, O.A. Nikitina // Innovative Information Technologies: Materials of the International Scientific-Practical Conference, Prague, Czech Republic, April 21–25. – 2014. – Part 2. – P. 452–457.

24. Баландин, М.Ю. Методы решения СЛАУ большой размерности / М.Ю. Баландин, Э.П. Шурина. – Новосибирск: НГТУ, 2000. – 65 с.

25. Турчак, Л.И. Основы численных методов / Л.И. Турчак, П.В. Плотников. – 2-е изд., перераб. и доп. – М.: Физматлит, 2002. – 304 с.

26. Вержбицкий, В.М. Численные методы (линейная алгебра и нелинейные уравнения): учеб пособие / В.М. Вержбицкий. – М: Директ-Медиа, 2013. – 432 с.

27. Golub, Gene H. Matrix Computations: 3rd Ed. / Gene H. Golub, Charles F. Van Loan. – Baltimore, MD, USA: Johns Hopkins University Press, 1996. – 694 pp.

28. Амосов, А.А. Вычислительные методы для инженеров: учеб. пособие / А.А. Амосов, Ю.А. Дубинский, Н.В. Копченова. – М.: Высшая школа, 1994. – 544 с.

29. Райс, Дж. Матричные вычисления и математическое обеспечение: пер. с англ. / Дж. Райс. – М.: Мир, 1984. – 264 с.

30. Каханер, Д. Численные методы и программное обеспечение / Д. Каханер, К. Моулер, С. Нэш. – М.: Мир, 1998. – 575 с.

31. Carpes, W.P. Analysis of the coupling of an incident wave with a wire inside a cavity using an FEM in frequency and time do-

mains / W.P. Carpes, L. Pichon and A. Razek // IEEE Trans. on Electromagn. Compat. – August 2002. – V. 44(3). – P. 470–475.

32. The OpenMP API specification for parallel programming [Электронный ресурс]. – Режим доступа: http://openmp.org/.

33. Open MPI: Open Source High Performance Computing [Электронный ресурс]. – Режим доступа: http://www.open-mpi.org/.

34. Параллельные вычисления CUDA [Электронный ресурс]. – Режим доступа: http://www.nvidia.ru/object/cuda-parallel-computing-ru.html.

35. Использование видеокарт для вычислений [Электронный ресурс]. – Режим доступа: http://www.gpgpu.ru/.

36. International Organization for Standardization, Geneva. Information technology – Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API) [C Language]. – December 1996.

37. Windows API [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/cc433218.

38. LAPACK – Linear Algebra PACKage [Электронный ресурс]. – Режим доступа: http://www.netlib.org/lapack/.

39. Automatically Tuned Linear Algebra Software (ATLAS) [Электронный ресурс]. – Режим доступа: http://math-atlas.sourceforge.net/.

40. Intel Math Kernel Library (Intel MKL) [Электронный ресурс]. – Режим доступа: https://software.intel.com/en-us/intel-mkl.

41. BLAS (Basic Linear Algebra Subprograms) [Электронный ресурс]. – Режим доступа: http://www.netlib.org/blas/.

42. Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms [Электронный ресурс]. – Режим доступа: http://eigen.tuxfamily.org/index.php.

43. Фаддеев, Д.К. Вычислительные методы линейной алгебры / Д.К. Фаддеев, В.Н. Фаддеева // Записки науч. семинара ЛОМИ. – 1975. – Т. 54. – С. 3–228.

44. Hoffman, Joe D. Numerical Methods for Engineers and Scientists, Second Edition / Joe D. Hoffman. – New York: CRC Press, 2001. – 823 p.

45. Higham, Nicholas J. Gaussian elimination / Nicholas J. Higham // Wiley Interdisciplinary Reviews: Computational Statistics. – 2011. – V. 3 (3). – P. 230–238.

46. Lam, Lay-Yong. Methods of solving linear equations in traditional China / Lam Lay-Yong and Shen Kangshen // Historia Mathematica. – 1989. – V. 16 (2). – P. 107–122.

47. Калиткин, Н.Н. Численные методы / Н.Н. Калиткин. – М: Наука, 1978. – 512 с.

48. Форсайт, Дж. Численное решение систем линейных алгебраических уравнений / Дж. Форсайт, К. Молер – М.: Мир, 1969. – 167 с.

49. Богачев, К.Ю. Практикум на ЭВМ. Методы решения линейных систем и нахождения собственных значений: учеб. пособие / К.Ю. Богачев. – М.: МГУ, 1998. – 137 с.

50. Saad, Y. Iterative methods for sparse linear systems (2nd ed.) / Y. Saad. – Second edition. – Philadelphia, PA, USA: SIAM, 2003. – 547 p.

51. Левитин, А. Алгоритмы: введение в разработку и анализ: пер. с англ. / А. Левитин. – М. : Вильямс, 2006. – 576 с.

52. Automatic Blocking of QR and LU Factorizations for Locality / Q. Yi, K. Kennedy, H. You, S. Keith, J. Dongarra // Proceedings of the 2004 Workshop on Memory System Performance, New York, NY, USA. – 2004. – P. 12–22.

53. Li, N. Crout Versions of ILU for General Sparse Matrices / N. Li, Y. Saad, E. Chow // SIAM Journal on Scientific Computing. – 2003. – V. 25(2). – P. 716–728.

54. GPU-Accelerated Parallel Sparse LU Factorization Method for Fast Circuit Analysis / K. He, S.X. Tan, H. Wang, G. Shi // Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. – 2015. – №. 99. – P. 1–10.

55. Pernice, M. A Multigrid-Preconditioned Newton-Krylov Method for the Incompressible Navier-Stokes Equations / M. Pernice, M.D. Tocci // SIAM Journal on Scientific Computing. – 2001. – V. 23(2). – P. 398–418.

56. Крукиер, Л.А. Обзор методов подпространства Крылова / Л.А. Крукиер, О.А. Пичугина, Л.Г. Чикина // XIV Международная конференция-школа с международным участием «Современные

проблемы математического моделирования». – Ростов н/Д: Изд-во Южного федерального университета, 2011. – С. 203–243.

57. Крылов, А.Н. О численном решении уравнения, которым в технических вопросах определяются частоты малых колебаний материальных систем / А.Н. Крылов // Известия Академии наук СССР. VII серия. Отделение математических и естественных наук. – 1931. – № 4. – С. 491–539.

58. 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.

59. Chow, E. Approximate inverse preconditioners via sparse-sparse iterations / E. Chow, Y. Saad // SIAM J. Sci. Comput. – 1998. – V. 19. – P. 995–1023.

60. Kaporin, I.E. New convergence results and preconditioning strategies for the conjugate gradient method / I.E. Kaporin // Linear Algebra Appl. – 1994. – V. 1. – P. 179–210.

61. Grote, M.J. Thomas Parallel Preconditioning with Sparse Approximate Inverses / M.J. Grote, T. Huckle // SIAM J. Sci. Comput. – 1997. – V. 18(3). – P. 838–853.

62. Benzi, M. A sparse approximate inverse preconditioner for the conjugate gradient method / M. Benzi, C.D. Meyer, M. Tuma // SIAM J. Sci. Comput. – 1996. – V. 17. – P. 1135–1149.

63. Benzi, M. A comparative study of sparse approximate inverse preconditioners / M. Benzi, M. Tuma // Appl. Numer. Math. – 1999. – V. 30. – P. 305–340.

64. Kolotilina, L.Yu. Factorized sparse approximate inverse preconditioning I. Theory / L. Yu. Kolotilina and A.Yu. Yeremin // SIAM J. Matrix Anal. Appl. – 1993. – V. 14. – P. 45–58.

65. Parallel Sparse Approximate Inverse Preconditioning on Graphic Processing Units / M.M. Dehnavi, D.M. Fernandez, J. Gaudiot, D.D. Giannacopoulos // Parallel and Distributed Systems, IEEE Transactions on. – 2013. – V. 24(9). – P. 1852–1862.

66. Mori, H. An ILU(p)-Preconditoner Bi-CGStab Method for Power Flow Calculation / H. Mori, F. Iizuka // Power Tech, 2007 IEEE Lausanne. – 2007. – P. 1474–1479.

67. Колотилина, Л.Ю. Явно предобусловленные системы линейных алгебраических уравнений с плотной матрицей / Л.Ю. Колотилина // Советская математика. – 1988. – № 43. – С. 2566–2573.

68. Куксенко, С.П. Совершенствование способов предфильтрации для решения СЛАУ с плотной матрицей итерационными методами с предобусловливанием в задачах вычислительной электродинамики / С.П. Куксенко, Т.Р. Газизов // Электромагнитные волны и электронные системы. – 2007. – № 9. – С. 12–17.

69. Gatsis, J. Preconditioning Techniques for a Newton–Krylov Algorithm for the Compressible Navier–Stokes Equations / J. Gatsis // The thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy Graduate. Department of Institute for Aerospace Studies University of Toronto. – 2013. – P. 144.

70. Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differentialand integral operators / C. Lanczos // J. Res. Nat. Bur. Stand. – 1950. – Vol. 45. – P. 255–282.

71. Arnoldi, W. The principle of minimized iterations in the solution of the matrix eigenvalue problem / W. Arnoldi // Quarterly of Applied Mathematics. – 1951. – V. 9. – P. 17–29.

72. Hestenes, M. Methods of conjugate gradients for solving linear systems / M. Hestenes, E. Stiefel // J. Res. Nat. Bur. Stand. – 1952. – Vol. 49. – P. 409–436.

73. Lanczos, C. Solution of systems of linear equations by minimized iterations / C. Lanczos // Journal of Research of the National Bureau of Standards. – 1952. – V. 49. – P. 33–53.

74. Paige, C. Solution of sparse indefinite systems of linear equations / C. Paige, M. Saunders // SIAM Journal on Numerical Analysis. –1975. – Vol. 12. – P. 617–629.

75. Paige, C. Solution of sparse indefinite systems of linear equations / C. Paige, M. Saunders // SIAM Journal on Numerical Analysis. – 1975. – V. 12. – P. 617–629.

76. Fletcher, R. Conjugate gradient methods for indefinite systems / R. Fletcher // Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974 / ed. by G. Watson, Springer Verlag, New York. – 1975. – P. 73–89.

77. Concus, P. A generalized conjugate gradient method for nonsymmetric systems of linear equations / P. Concus, G. Golub // Computer methods in Applied Sciences and Engineering, Second International Symposium / edited by R. Glowinski and J. Lions, Springer Verlag. – New York, 1976. – December. – P. 56–65.

78. Vinsome, P. ORTHOMIN: an iterative method for solving sparse sets of simultaneous linear equations / P. Vinsome // Proceedings of the Fourth Symposium of Reservoir Simulation, Society of Petroleum Engineers of AIME. – 1976. – P. 149–159.

79. Meijerink, J.A. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix / J.A. Meijerink, H.A. van der Vorst // Mathematics of Computation. – 1977. – V. 31(137). – P. 148–162.

80. Widlund, O. A Lanczos method for a class of non-symmetric systems of linear equations / O. Widlund // SIAM J. Numer. Anal. – 1978. – V. 15. – P. 801–802.

81. Jea, K. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods / K. Jea, D. Young // Linear Algebra and its Applications. – 1980. – V. 34. – P. 159–194.

82. Saad, Y. Krylov subspace methods for solving large unsymmetric linear systems / Y. Saad // Mathematics of Computation. – 1981. – V. 37. – P. 105–126.

83. Paige, C. LSQR: an algorithm for sparse linear equations and sparse least squares / C. Paige, M. Saunders // ACM Transactions on Mathematical Software. – 1982. – V. 8. – P. 43–71.

84. Eisenstat, S.C. Variational iterative methods for nonsymmetric systems of linear equations / S.C. Eisenstat, H.C. Elman, M.H. Schultz // SIAM Journal on Numerical Analysis. – V. 20(2). – P. 345–357.

85. Saad, Y. GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems / Y. Saad, M.H. Schultz // SIAM Journal on Scientific and Statistical Computing. – 1986. – V. 7. – P. 856–869.

86. Sonneveld, P. CGS, a fast Lanczos-type solver for nonsymmetric linear systems / P. Sonneveld. // SIAM J. Sci. Statist. Comput. – 1982. –V. 10. – P. 36–52.

87. Freund, R.W. QMR: a quasi-minimal residual method for non-Hermitian linear systems / R.W. Freund, N.M. Nachtigal // Numer. Math. – 1991. – V. 60. – P. 315–339.

88. Van der Vorst, H.A. BI-CGSTAB: A Fast and Smoothly Converging Variant of BI-CG for the Solution of Nonsymmetric Linear Systems / H.A. Van der Vorst // SIAM J. Sci. Stat. Comput. – 1992. – V. 13(2) – P. 631–644.

89. Gutknecht, M.H. Variants of BICGSTAB for matrices with complex spectrum / M.H. Gutknecht // SIAM J. Sci. Comput. – 1993. – V. 14. – P. 1020–1033.

90. Sleijpen, G.L. BiCGStab(l) for linear equations involving unsymmetric matrices with complex spectrum / G.L. Sleijpen, D.R. Fokkema // Electronic Transactions on Numerical Analysis. – 1993. – V. 1. – P. 11–32.

91. Freund, R.W. An implementation of the QMR method based on coupled two-term recurrences / R.W. Freund, N.M. Nachtigal // SIAM Journal on Scientific Computing. – 1994. – Vol. 15. – P. 313–337

92. Weiss, R. Error-minimizing Krylov subspace methods / R. Weiss // SIAM Journal on Scientific Computing. – 1994. – V. 15. – P. 511–527.

93. A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems / T.F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, C.H. Tong // SIAM Journal on Scientific Computing. – 1994. – V. 15. – P. 338–347.

94. Kasenally, E.M. GMBACK: a generalized minimum backward error algorithm for nonsymmetric linear systems / E.M. Kasenally // SIAM J. Sci. Comput. – 1995. – V. 16. – P. 698–719.

95. Fokkema, D.R. Generalized conjugate gradient squared / D.R. Fokkema, G.L.G. Sleijpen, H.A. van der Vorst // Journal of Computational and Applied Mathematics. – 1996. – V. 71. – P. 125–146.

96. De Sturler, E. Truncation strategies for optimal Krylov subspace methods / E. de Sturler // SIAM Journal on Numerical Analysis. – 1999. – V. 36(3). – P. 864–889.

97. Писсанецки, С. Технология разреженных матриц : пер. с англ. / С. Писсанецки. – М.: Мир, 1988. – 410 с.

98. Alvarado, F.L. A note on sorting sparse matrices / F.L. Alvarado // Proceedings of the IEEE. – 1979. – V. 67(9). – P. 1362–1363.

99. Тьюарсон, Р. Разреженные матрицы / Р. Тьюарсон ; пер. с англ. Э.М. Пейсаховича, под ред. Х.Д. Икрамова. – М.: Мир, 1977. – 172 с.

100. Knuth, D. The Art of Computer Programming: Fundamental algorithms / D. Knuth. – Addison-Wesley, 1968. – 634 p.

101. Rheinboldt, W.C. Programs for the Solution of Large Sparse Matrix Problems Based on the Arc-Graph Structure: Technical Report TR-262 / W.C. Rheinboldt, C.K. Mesztenyi. – Computer Science Center, University of Maryland, College Park MD, 1973.

102. Larcombe, M. A hst processing approach to the solution of large sparse sets of matrix equations and the factorization of the overall matrix / M. Larcombe // Large Sparse Sets of Linear Equations / Ed. Reid, J.K. – London: Academm Press, 1971.

103. Sherman, A.H. On the Efficient Solution of Sparse Systems of Linear and Nonlinear Equations: Ph.D. dissertation. – Department of Computer Science, Yale University, 1975.

104. Chang, A. Apphcatton of sparse matrix methods in electric power system analysis / A. Chang // Sparse Matrix: Proceedings / Ed. R. Willoughby, IBM Watson Research Center, RAl1707. – 1969, March. – P. 113–122.

105. Gustavson, F.G. Some basic techniques for solving sparse systems of linear equations in Sparse Matrices and Their Applications / F.G. Gustavson ; Eds. D.J.Rose & R.A. WiHougby. – New York: Plenum Press, 1972. – P. 41–52.

106. Saad, Y. Numerical solution of large nonsymmetric eigenvalue problems / Y. Saad // Comp. Phys. Comm. – 1989. – V. 53. – P. 71–90.

107. LINPACK Users' Guide / J.J. Dongarra, J. R. Bunch, C.B. Moler, G.W. Stewart // SIAM, Philadelphia. – 1979.

108. Tinetti, F. Performance of Scientific Processing in NOW: Matrix Multiplication Example / F. Tinetti // Journal of Computer Science and Technology. – 2001. – V. 1(4). – P. 78–87.

109. Sparse Matrix Storage Formats [Электронный ресурс]. – Режим доступа: http://www.netlib.org/utk/people/JackDongarra/etemplates/node372. html.

110. Jiang, J. The spatial relationship of DCT coefficients between a block and its sub-blocks / J. Jiang, G. Feng // Signal Processing, IEEE Transactions on. – 2002. – V. 50(5). – P. 1160–1169.

111. Improving the Performance of the Sparse Matrix Vector Product with GPUs / F. Vazquez, G. Ortega, J.J. Fernandez, E.M. Garzon // Computer and Information Technology (CIT), IEEE 10th International Conference on. – 2010. – P. 1146–1151.

112. Kestur, S. Towards a Universal FPGA Matrix-Vector Multiplication Architecture / S. Kestur, J.D. Davis, E.S. Chung // Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE: 20th Annual International Symposium on. – 2012. – P. 9–16.

113. Karimi, S. A New Iterative Solution Method for Solving Multiple Linear Systems / S. Karimi // Advances in Linear Algebra & Matrix Theory. – 2012. – V. 2(3). – P. 25–30.

114. Yu, F. Recursive least-squares algorithms with good numerical stability for multichannel active noise control / F. Yu, M. Bouchard // Acoustics, Speech, and Signal Processing, 2001: Proceedings. (ICASSP '01). 2001 IEEE International Conference on. – 2001. – V. 5. – P. 3221–3224.

115. Paige, C.C. LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares / C.C. Paige, M.A. Saunders // ACM Transactions on Mathematical. – 1982. – V. 8(1). – P. 43–71.

116. Kress, R. Linear Integral Equations: Springer-Verlag / R. Kress. – New York, 1989.

117. Jain, Anil K. Fundamentals of Digital Image Processing / K. Anil Jain // Englewood Cliffs. – NJ: Prentice Hall, 1989.

118. Calgaro, C. Incremental incomplete LU factorizations with applications to time-dependent PDEs / C. Calgaro, J.P. Chehab, Y. Saad // Numer. Lin. Algebra with Appl. – 2010. – № 17(5). – P. 811–837.

119. Simoncini, V. A hybrid block GMRES method for nonsymmetric systems with multiple right-hand sides / V. Simoncini and E. Gallopoulos // Journal of Computational and Applied Mathematics. – 1996. – V. 66. – P. 457–469.

120. Toutounian, F. Global least squares method (Gl-LSQR) for solving general linear systems with several right-hand sides / F. Toutounian, S. Karimi // Applied Mathematics and Computation. – 2006. – V. 178(2). – P. 452–460.

121. Simoncini, V. An Iterative Method for Nonsymmetric Systems with Multiple Right-hand Sides / V. Simoncini, E. Gallopoulos // SIAM J. Sci. Comput. – 1995. – V. 16(4). – P. 917–933.

122. Liu Yang. An efficient method MEGCR for solving systems with multiple right-hand sides in 3-D parasitic inductance extraction / Liu Yang, Xiaobo Guo, Zeyi Wang // Design Automation Conference, 2004: Proceedings of the ASP-DAC 2004. Asia and South Pacific. – 2004. – P. 702–706.

123. Суровцев, Р.С. Ускорение многократного решения СЛАУ с частично изменяющейся матрицей / Р.С. Суровцев, С.П. Куксенко, Т.Р. Газизов // Доклады Томского государственного университета систем управления и радиоэлектроники. – 2011. – № 2(24), ч. 1. – С. 141–144.

124. Суровцев, Р.С. Исследование ускорения многократного решения СЛАУ с частично изменяющейся матрицей блочным методом / Р.С. Суровцев, В.К. Салов // Электромагнитные волны и электронные системы. – 2012. – Т. 17(10). – С. 22–24.

125. Суровцев, Р.С. Вычисление матрицы емкостей произвольной системы проводников и диэлектриков методом моментов: зависимость ускорения за счет блочного LU-разложения от порядка матрицы СЛАУ / Р.С. Суровцев, С.П. Куксенко // Известия высших учебных заведений. Физика. – 2012. – Т. 55 (9/3). – С. 126–130.

126. Gazizov, T.R. Analytic expressions for MOM calculation of capacitance matrix of two dimensional system of conductors and dielectrics having arbitrarily oriented boundaries / T.R. Gazizov // Proc. of the 2001 IEEE EMC Symposium, Montreal, Canada, August 13–17. – 2001. – Vol. 1. – P. 151–155.

127. Форматы хранения разреженных матриц и ускорение решения СЛАУ с плотной матрицей итерационными методами / Р.Р. Ахунов, С.П. Куксенко, В.К. Салов, Т.Р. Газизов // Численные методы и вопросы организации вычислений: XXV Записки науч. семинара ПОМИ. – 2012. – Т. 405(25). – С. 24–39.

128. Усовершенствование алгоритма ILU(0)-разложения, использующего разреженный строчный формат / Р.Р. Ахунов, С.П. Куксенко, В.К. Салов, Т.Р. Газизов // Численные методы и вопросы организации вычислений: XXV Записки науч. семинара ПОМИ. – 2012. – Т. 405(25). – С. 40–53.

129. Gibson, W.C. The method of moments in electromagnetics / W.C. Gibson. – Boca Raton, FL: Taylor & Francis Group, 2007. – P. 272.

130. Газизов, Т.Р. Оптимизация допуска обнуления при решении СЛАУ итерационными методами с предобусловливанием в задачах вычислительной электродинамики / Т.Р. Газизов, С.П. Куксенко // Электромагнитные волны и электронные системы. – 2004. – № 8. – С. 26–28.

131. Многократное решение СЛАУ с частично изменяющейся матрицей итерационным методом / Р.Р. Ахунов, С.П. Куксенко, В.К. Салов, Т.Р. Газизов // Численные методы и вопросы организации вычислений: XXV Записки науч. семинара ПОМИ. – 2013. – Т. 419. – С. 16–25.

132. Ахунов, Р.Р. Ускорение многократного решения СЛАУ итерационным методом при вычислении емкости микрополосковой линии в широком диапазоне изменения ее размеров / Р.Р. Ахунов, С.П. Куксенко, Т.Р. Газизов // Численные методы и вопросы организации вычислений: XXVII Записки науч. семинара ПОМИ. – 2014. – Т. 428. – С. 32–41.

133. Kuksenko, S.P. Dense linear system solution by preconditioned iterative methods in computational electromagnetics / S.P. Kuksenko, T.R. Gazizov // 19th Int. Zurich Symp. Electromagn. Compatibility. – 2008. – P. 918–921.

134. Ахунов, Р.Р. Многократное решение СЛАУ итерационным методом с переформированием матрицы предобусловливания / Р.Р. Ахунов, С.П. Куксенко, Т.Р. Газизов // Численные методы и вопросы организации вычислений: XXVII Записки науч. семинара ПОМИ. – 2014. – Т. 428. – С. 42–48.

135. Ахунов, Р.Р. Ускорение многократного решения СЛАУ с изменяющейся матрицей / Р.Р. Ахунов, С.П. Куксенко // Международная конференция «Актуальные проблемы вычислительной и прикладной математики 2015», посвященная 90-летию со дня ро-

ждения академика Гурия Ивановича Марчука, Новосибирск, 19–23 октября. – 2015. – С. 84–90.

136. Improved design of modal filter for electronics protection / A.M. Zabolotsky, T.R. Gazizov, A.O. Melkozerov, P.E. Orlov, E.S. Dolganov // Proc. of 31-th Int. conf. on lightning protection, Vienna, Austria. September 2–7. – 2012.

137. Вержбицкий, В.М. Основы численных методов / В.М. Вержбицкий. – М.: Высшая школа, 2002. – 840 с.

138. Gazizov, T.R. Acceleration of Multiple Solution of Linear Systems for Analyses of Microstrip Structures / T.R. Gazizov, S.P. Kuksenko, R.R. Akhunov // International journal of mathematical models and methods in applied sciences. – 2015. – Vol. 9. – P. 721–726.

139. Заболоцкий, А.М. Использование зеркальной симметрии для совершенствования модальной фильтрации / А.М. Заболоцкий // Доклады Томского государственного униваерситета систем управления и радиоэлектроники. – 2015. – № 2(36). – С. 41–44.

140. Ахунов, Р.Р. Простой способ ускорения вычисления емкостных матриц полосковой структуры при изменении ее геометрического параметра / Р.Р. Ахунов, С.П. Куксенко, Т.Р. Газизов // Доклады Томского государственного униваерситета систем управления и радиоэлектроники. – 2015. – № 4. – С 144–148.

Оглавление

Научное издание

Ахунов Роман Раисович
Куксенко Сергей Петрович
Газизов Тальгат Рашитович
Орлов Павел Евгеньевич

МНОГОКРАТНОЕ РЕШЕНИЕ СИСТЕМ
ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ
ИТЕРАЦИОННЫМИ МЕТОДАМИ С ПРЕДОБУСЛОВЛИВАНИЕМ
В ЗАДАЧАХ ЭЛЕКТРОМАГНИТНОЙ СОВМЕСТИМОСТИ

Монография

Подписано в печать 02.12.15. Формат 60x84/16.
Усл.-печ. л. 8,84. Тираж 100 экз. Заказ 896.

Томский государственный университет
систем управления и радиоэлектроники.
634050, г. Томск, пр. Ленина, 40. Тел. (3822) 533018.