

А. Н. Чурилов, А. В. Гессен

ИССЛЕДОВАНИЕ ЛИНЕЙНЫХ МАТРИЧНЫХ НЕРАВЕНСТВ

ПУТЕВОДИТЕЛЬ ПО ПРОГРАММНЫМ ПАКЕТАМ

```
sz=size(A);
setlmis([ ]);
H=lmis([H A(+,A'H) Hb]);
theta=lmivar(1,[1
b'H 0]);
MLMI=newlmi;
lmiterm([MLMI 1 1 1],theta);
lmiterm([MLMI 1 2 1],H_iA_i+A'iHi);
lmiterm([MLMI 1 2 2],H_iBi);
lmiterm([MLMI 2 2 1],B'_tH);
lmiterm([-MLMI 1 2 0],-0.5*c);
lmiterm([-MLMI 2 2 0],tau);
Hpos=newlmi;
lmiteh([-Hpos 1 A'_lambda H - Hb]);
lmisys=getlmis;
[tmin,x0] blasp(lmis,0);
[ - p1 p2 c c' ] <= [ - 1/2(mu_1 + mu_2)c' ]
```

$$H + \tau cc' \geq 0, \quad \tau > 0$$

А. Н. Чурилов, А. В. Гессен

ИССЛЕДОВАНИЕ ЛИНЕЙНЫХ
МАТРИЧНЫХ НЕРАВЕНСТВ

ПУТЕВОДИТЕЛЬ ПО ПРОГРАММНЫМ ПАКЕТАМ

Издательство С.-Петербургского университета

2004

УДК 004.42, 519.852.2

ББК 32.973.26-018.2

Ч-93

Рецензенты: д-р физ.-мат. наук, проф. А. Х. Гелиг (СПбГУ),
д-р пед. наук, проф. Г. Г. Хамов (РГПУ им. А.И. Герцена)

*Печатается по постановлению
Редакционно-издательского совета
математико-механического факультета
Санкт-Петербургского государственного университета*

Чурилов А. Н., Гессен А. В.

Ч-93 **Исследование линейных матричных неравенств. Путеводитель по программным пакетам.** — СПб.: Изд-во С.-Петербургского ун-та, 2004. — 148 с.
ISBN 5-288-03585-7

Книга содержит описание современных программных средств исследования линейных матричных неравенств. Подробно рассматриваются программные пакеты LMILab, LMITOOL, SeDuMi Interface, YALMIP и KYPD, которые являются пакетами расширения систем программирования MATLAB и Scilab.

Книга предназначена для специалистов в области теории управления, механики, математического моделирования, интересующихся применением систем MATLAB и Scilab и аппарата линейных матричных неравенств.

ББК 32.973.26-018.2

ISBN 5-288-03585-7

© А. Н. Чурилов, А. В. Гессен, 2004

Оглавление

| | |
|---|-----------|
| Предисловие | 6 |
| Глава 1. Общие сведения | 8 |
| 1.1. Задачи исследования линейных матричных неравенств | 8 |
| 1.2. Нестрогие линейные матричные неравенства | 11 |
| 1.3. Двойственность | 11 |
| 1.4. Численные методы исследования линейных матричных неравенств | 13 |
| 1.5. Матричные переменные | 17 |
| 1.6. Формулы Шура | 19 |
| 1.7. Некоторые примеры | 21 |
| 1.8. <i>S</i> -процедура | 23 |
| 1.9. Решатели и интерфейсные пакеты | 24 |
| 1.10. Обзор интерфейсных пакетов исследования LMI | 27 |
| 1.10.1. Интерфейсы для работы с одним решателем | 27 |
| 1.10.2. Интерфейсы для работы с несколькими решателями | 30 |
| 1.10.3. Интерфейсы — надстройки над другими интерфейсами | 31 |
| 1.11. История метода линейных матричных неравенств в теории управления | 32 |
| Глава 2. Пакет LMILab | 34 |
| 2.1. Создание системы линейных матричных неравенств | 34 |
| 2.1.1. Функция <code>setlmis</code> | 35 |
| 2.1.2. Функция <code>lmivar</code> | 36 |
| 2.1.3. Функция <code>newlmi</code> | 39 |
| 2.1.4. Функция <code>lmiterm</code> | 39 |
| 2.1.5. Функция <code>getlmis</code> | 42 |
| 2.1.6. Интерактивное создание системы линейных матричных неравенств. Команда <code>lmiedit</code> | 42 |
| 2.2. Получение информации о системе | 45 |

| | |
|---|-----------|
| 2.2.1. Функция <code>lminbr</code> | 45 |
| 2.2.2. Функция <code>matnbr</code> | 45 |
| 2.2.3. Функция <code>decnbr</code> | 45 |
| 2.2.4. Команда <code>lmiinfo</code> | 45 |
| 2.2.5. Команда <code>decinfo</code> | 48 |
| 2.3. Взаимные преобразования матричных и скалярных переменных | 49 |
| 2.3.1. Функция <code>dec2mat</code> | 49 |
| 2.3.2. Функция <code>mat2dec</code> | 50 |
| 2.3.3. Функция <code>defcx</code> | 50 |
| 2.4. Исследование системы линейных матричных неравенств | 52 |
| 2.4.1. Функция <code>feasp</code> | 53 |
| 2.4.2. Функция <code>mincx</code> | 54 |
| 2.4.3. Функция <code>gevp</code> | 55 |
| 2.4.4. Функция <code>basiclmi</code> | 57 |
| 2.5. Оценивание линейных матричных неравенств | 57 |
| 2.5.1. Функция <code>evallmi</code> | 57 |
| 2.5.2. Функция <code>showlmi</code> | 58 |
| 2.6. Модификация системы линейных матричных неравенств | 58 |
| 2.6.1. Функция <code>dellmi</code> | 58 |
| 2.6.2. Функция <code>delmvar</code> | 58 |
| 2.6.3. Функция <code>setmvar</code> | 59 |
| 2.7. Примеры работы с пакетом | 59 |
| 2.7.1. Круговой критерий абсолютной устойчивости | 59 |
| 2.7.2. Критерий Попова абсолютной устойчивости | 62 |
| 2.7.3. Оценка запаса устойчивости линейной системы | 64 |
| 2.7.4. Оценка, связанная с неравенством Ляпунова | 65 |
| Глава 3. Пакет LMITOOL | 67 |
| 3.1. Общая характеристика пакета | 67 |
| 3.2. Описание функций пакета | 69 |
| 3.2.1. Функция <code>lmisolver</code> | 69 |
| 3.2.2. Функция <code>lmitool</code> (неинтерактивный режим) | 70 |
| 3.2.3. Функция <code>lmitool</code> (интерактивный режим) | 71 |
| 3.3. Примеры работы с пакетом | 73 |
| 3.3.1. Круговой критерий абсолютной устойчивости | 73 |
| 3.3.2. Критерий Попова | 76 |
| 3.3.3. Проверка условий пассивности | 77 |
| 3.3.4. Классическая задача линейного программирования | 79 |
| 3.4. Пакет LMITOOL в системе MATLAB | 80 |

| | |
|---|------------|
| Глава 4. Пакет SeDuMi Interface | 81 |
| 4.1. Общая характеристика пакета | 81 |
| 4.2. Создание оптимизационной задачи | 83 |
| 4.2.1. Функция <code>sdmpb</code> | 83 |
| 4.2.2. Функция <code>sdmvar</code> | 84 |
| 4.2.3. Задание матричных ограничений | 89 |
| 4.2.4. Функция <code>sdmobj</code> | 98 |
| 4.3. Решение оптимизационной задачи | 99 |
| 4.3.1. Функция <code>sdmsol</code> | 100 |
| 4.3.2. Регулировка параметров решателя | 102 |
| 4.4. Модификация задачи: функция <code>sdmset</code> | 106 |
| 4.5. Получение информации о задаче: функция <code>get</code> | 108 |
| 4.6. Примеры использования пакета | 111 |
| 4.6.1. Круговой критерий абсолютной устойчивости | 111 |
| 4.6.2. Оценка коэффициента диссипативности | 113 |
| Глава 5. Пакет YALMIP | 116 |
| 5.1. Общая характеристика пакета | 116 |
| 5.2. Описание матричных переменных | 117 |
| 5.3. Создание системы ограничений | 120 |
| 5.4. Решение задачи полуопределенного программирования | 123 |
| 5.5. Примеры исследования линейных матричных неравенств | 126 |
| 5.5.1. Круговой критерий абсолютной устойчивости | 126 |
| 5.5.2. Оценка H_∞ -нормы системы управления | 127 |
| 5.6. Использование ограничений типа билинейных матричных неравенств | 128 |
| 5.7. Вспомогательные команды | 132 |
| Глава 6. Пакет KYPD | 133 |
| 6.1. Общая характеристика пакета | 133 |
| 6.2. Описание основных функций | 134 |
| 6.3. Работа с KYPD в рамках пакета YALMIP | 137 |
| Приложение 1 | 139 |
| Приложение 2 | 141 |
| Литература | 142 |
| Указатель команд и функций | 148 |

Предисловие

Эта книга содержит обзор основных программных средств исследования линейных матричных неравенств и руководства пользователя популярных программных пакетов, служащих этой цели, — LMILab, LMITOOL, SeDuMi Interface, YALMIP и KYPD. Она рассчитана в основном не на специалистов в области методов оптимизации, а на людей, интересующихся приложениями этих методов в теории управления, механике, электротехнике и других областях. Они имеют возможность, не вдаваясь в математические подробности, получить готовый инструмент для своих исследований.

Одним из наиболее распространенных средств моделирования технических систем является система MATLAB фирмы The Mathworks, Inc. Она состоит из ядра и дополнительных пакетов программ (тубоксов). В настоящее время на русском языке имеется обширная литература как по системе MATLAB в целом, так и по ее отдельным пакетам (см., например, [9, 14, 15, 21, 22]). Подробную библиографию и сведения о методических разработках, посвященных MATLAB, можно найти на сайте <http://www.exponenta.ru>. Документация на английском языке доступна на сайте компании The Mathworks, Inc. <http://www.mathworks.com>. Вопросам моделирования систем управления с помощью MATLAB посвящены монографии [2, 3].

Другая популярная система моделирования, Scilab, была создана во Франции, в Научно-исследовательском институте информатики и автоматики (INRIA) в рамках исследовательского проекта META 2. Scilab является некоммерческой разработкой, поддерживаемой французским правительством. В качестве преимуществ системы Scilab можно отметить следующие. Во-первых, как сам пакет, так и документация к нему, распространяются совершенно бесплатно. Их можно найти на сайте INRIA по адресу <http://scilabsoft.inria.fr> или на сайте <http://www.scilab.org>. Объем архива с дистрибутивом Scilab версии 3.0 составляет около 10 Mb. Во-вторых, Scilab весьма нетребователен к системным ресурсам, чем выгодно отличается от последних версий MATLAB. К сожалению, в России

Scilab еще мало распространен, известные нам публикации на эту тему на русском языке — глава в книге [3] и web-книга [17].

Перед тем, как приступить к чтению этой брошюры, читателям нужно получить сведения о среде разработки и языке программирования систем MATLAB или Scilab, используя общие справочные пособия (по крайней мере, что касается MATLAB, недостатка в них нет). Желательно также знакомство с пакетом Control System Toolbox системы MATLAB (например, с помощью справочника [15]). Также желательно (но не обязательно) знакомство с общими принципами объектно-ориентированного программирования, так как некоторые пакеты используют данные объектного типа.

Описанных в книге пакеты могут быть использованы для исследования различных вопросов анализа и синтеза систем управления [29, 35], динамики нейронных сетей [31, 49], а также расчетов в различных областях техники (многочисленные публикации с примерами таких расчетов можно найти на сайте группы С. Бойда из Стенфордского университета <http://www.stanford.edu/~boyd/>).

Глава 4 этой книги была написана А. В. Гессен, остальные главы — А. Н. Чуриловым.

Авторы благодарят рецензентов А. Х. Гелига и Г. Г. Хамова, а также Б. Р. Андриевского, С. В. Гусева, И. Н. Митрофанова, П. В. Пакшина, Н. В. Утину, А. Л. Фрадкова, А. И. Шепелявого и всех друзей и коллег, взявших на себя труд ознакомиться с рукописью книги и высказать свое мнение. Очень полезным оказалось обсуждение этой работы с нашими зарубежными коллегами Д. Поселлем (D. Peaucelle) и Л. Файбусовичем (L. Faybusovich).

Работа над этой книгой была частично поддержана РФФИ (проект 02-01-00542) и Советом по грантам Президента РФ для поддержки молодых российских ученых и ведущих научных школ РФ (проект НШ-2257.2003.1). Авторы выражают признательность за финансовую поддержку руководству математико-механического факультета, кафедры и лаборатории теоретической кибернетики Санкт-Петербургского государственного университета и научно-исследовательской части Санкт-Петербургского государственного морского технического университета.

Глава 1. Общие сведения

В этой главе при изложении основных определений и постановок мы в основном будем следовать монографии [35].

1.1. Задачи исследования линейных матричных неравенств

Рассмотрим отображение $\mathbb{R}^m \rightarrow \mathbb{R}^{n \times n}$ вида

$$F(x) = F_0 + \sum_{i=1}^m x_i F_i, \quad (1.1)$$

где F_i — вещественные симметричные $n \times n$ -матрицы, x_i — координаты вектора $x \in \mathbb{R}^m$. Элементы вектора x будем называть скалярными переменными. Очевидно

$$F(\alpha x + (1 - \alpha)y) = \alpha F(x) + (1 - \alpha)F(y)$$

при всех $x, y \in \mathbb{R}^m$, $\alpha \in \mathbb{R}$. Функция $F(x)$, обладающая таким свойством, называется аффинной.

Поскольку при всех x матричная функция $F(x)$ симметрична, имеет смысл рассматривать неравенства вида $F(x) > 0$ или $F(x) \geq 0$, понимая их как положительную или неотрицательную определенность матрицы $F(x)$. (Аналогично рассуждая, можно рассмотреть неравенства $F(x) < 0$ и $F(x) \leq 0$.) Неравенства такого типа называют линейными матричными неравенствами, в англоязычной терминологии обычно используется сокращение LMI (linear matrix inequalities). Из (1.1) очевидно вытекает свойство выпуклости:

$$F(\alpha x + (1 - \alpha)y) \leq \alpha F(x) + (1 - \alpha)F(y)$$

при всех $x, y \in \mathbb{R}^m$, $0 \leq \alpha \leq 1$. Одновременно справедливо неравенство

$$F(\alpha x + (1 - \alpha)y) \geq \alpha F(x) + (1 - \alpha)F(y),$$

поэтому выпуклой является и функция $-F(x)$. Отсюда, множества

$$\{x \mid F(x) \leq 0\}, \quad \{x \mid F(x) \geq 0\}$$

выпуклы, их граница является кусочно-гладкой.

Система нескольких линейных матричных неравенств

$$F^{(1)}(x) > 0, \dots, F^{(p)}(x) > 0$$

сводится к одному матричному неравенству для блочно-диагональной матрицы

$$\begin{bmatrix} F^{(1)}(x) & & \\ & \ddots & \\ & & F^{(p)}(x) \end{bmatrix} > 0$$

и не требует отдельного рассмотрения.

Простейшая задача, которую связывают с отображением $F(x)$, — задача разрешимости неравенства $F(x) > 0$ или $F(x) \geq 0$. (Ее английское название feasibility problem, сокращенно FEASP.) При этом требуется определить, существует ли решение линейного матричного неравенства, и, если оно существует, указать какое-либо из решений. Обычно численные алгоритмы не позволяют получить полное описание всего множества решений (которое очевидно выпукло).

Более общая постановка задачи разрешимости — одновременное рассмотрение неравенства и равенства:

$$F(x) > 0, \quad Ax = b,$$

где A — некоторая матрица, b — заданный n -вектор.

Следующая задача, связанная с отображением $F(x)$ — оптимизационная задача вида

$$\begin{aligned} c'x &\rightarrow \min, \\ F(x) &\geq 0 \end{aligned}$$

или

$$\begin{aligned} c'x &\rightarrow \min, \\ F(x) &\geq 0, \quad Ax = b. \end{aligned}$$

Здесь c — заданный m -вектор, штрих обозначает транспонирование. Эта задача — частный случай задачи выпуклого программирования (целевая функция линейна, ограничения задаются выпуклым отображением). В англоязычной литературе ее называют задачей полуопределенного программирования (semidefinite programming, сокращенно SDP или

SP). Очевидно классическая задача линейного программирования может быть сведена к задаче полуопределенного программирования. Обратное утверждение неверно (см., например, [64]). Таким образом, полуопределенное программирование занимает промежуточное место между линейным программированием и общим выпуклым программированием.

Еще одна оптимизационная задача выпуклого программирования имеет вид

$$\begin{aligned} \lambda &\rightarrow \min, \\ \lambda I_n - F(x) &> 0, \quad G(x) > 0. \end{aligned}$$

Ее называют также проблемой собственных чисел (eigenvalue problem, EVP). Здесь $F(x)$ и $G(x)$ — симметричные матрицы, аффинные по x , I_n — единичная $n \times n$ -матрица, λ — скалярная переменная.

Очевидно, если минимизировать значение λ при фиксированном значении x , то получим оценку наибольшего собственного значения матрицы $F(x)$. Когда x варьируется, получаем минимум наибольшего собственного значения матрицы $F(x)$ по всем x таким, что $G(x) > 0$. Поскольку матрица $\lambda I_n - F(x)$ аффинна по x и по λ , эта задача может рассматриваться как частный случай задачи полуопределенного программирования.

Естественное обобщение предыдущей задачи — обобщенная проблема собственных чисел (generalized eigenvalue problem, GEVP). Ее постановка следующая:

$$\begin{aligned} \lambda &\rightarrow \min, \\ \lambda E(x) - F(x) &> 0, \quad G(x) > 0. \end{aligned}$$

Здесь $E(x)$ — также симметричная матрица, аффинная по x . В данной задаче речь идет о наибольшем собственном значении матричного пучка $\lambda E(x) - F(x)$.

Наконец, отметим еще задачу вычисления наибольшего определителя (determinant maximization, MAXDET):

$$\begin{aligned} c'x + \ln \det G(x)^{-1} &\rightarrow \min, \\ F(x) &\geq 0, \quad G(x) > 0. \end{aligned}$$

Можно показать, что так определенная целевая функция является выпуклой.

Перечисленные постановки не исчерпывают различные задачи, связанные с линейными матричными неравенствами, которые описаны в литературе.

1.2. Нестрогие линейные матричные неравенства

В приложениях встречаются случаи как строгих матричных неравенств вида $F(x) > 0$, так и нестрогих $F(x) \geq 0$. В то же время, некоторые численные алгоритмы работают только со строгими матричными неравенствами. Однако эту трудность можно обойти — нестрогое неравенство всегда может быть сведено к строгому.

Если для нестрогого неравенства $F(x) \geq 0$ существует такой вектор x_0 , что $F(x_0) > 0$, то это нестрогое неравенство называется строго разрешимым. Множество решений строго разрешимого нестрогого неравенства получается как замыкание множества решений соответствующего строгого неравенства.

Однако встречаются случаи, когда нестрогое неравенство имеет решение, а строгое — нет. В этом случае можно сделать линейную замену скалярных переменных вида $x = Az + b$, где z — вектор скалярных переменных, имеющий размерность меньшую, чем x , а A , b — некоторые матричные коэффициенты, такую, что исходное неравенство $F(x) \geq 0$ эквивалентно некоторому неравенству $\tilde{F}(z) \geq 0$, которое либо неразрешимо, либо строго разрешимо. Ряд функций исследования линейных матричных неравенств (например, функция `lmisolver` пакета LMITOOL) выполняет такое преобразование (его подробное описание можно найти в [35]).

Рассмотрим задачу полуопределенного программирования

$$\begin{aligned} c'x &\rightarrow \min, \\ F(x) &\geq 0. \end{aligned}$$

Вектора x , удовлетворяющие ограничению $F(x) \geq 0$, называются допустимыми.¹ Они составляют допустимое множество оптимизационной задачи. Вектора x , для которых выполнено строгое неравенство $F(x) > 0$, называют строго допустимыми. Если множество строго допустимых векторов непусто, то говорят, что для оптимизационной задачи выполнено условие регулярности или условие Слейтера.

1.3. Двойственность

Поскольку полуопределенное программирование является частным случаем выпуклого программирования, для него справедливы все основ-

¹Заметим, что английскому термину «feasible» в русском языке могут отвечать разные термины — «допустимый» или «разрешимый», в зависимости от контекста.

ные результаты теории двойственности для выпуклых задач. Тем не менее, специфическая матричная структура задач полуопределенного программирования может быть учтена и эффективно использована. Подробное обсуждение вопросов двойственности в полуопределенном программировании с большим количеством примеров и ссылок содержится в обзорной статье [64]. Приведем некоторые формулировки из этой работы.

Рассмотрим оптимизационную задачу

$$\begin{aligned} c'x &\rightarrow \min, \\ F(x) = F_0 + \sum_{i=1}^m x_i F_i &\geq 0, \end{aligned} \tag{1.2}$$

которую будем называть прямой. Матричная функция

$$\mathcal{A}(x) = \sum_{i=1}^m x_i F_i$$

определяет линейное отображение из пространства \mathbb{R}^m в линейное пространство вещественных симметричных $n \times n$ -матриц \mathcal{S} . В пространстве \mathcal{S} можно ввести скалярное произведение матриц A и B с помощью формулы $\text{tr}(AB)$. Здесь tr — след матрицы (сумма ее диагональных элементов). Тогда двойственной для задачи (1.2) является следующая задача относительно неизвестной симметричной $n \times n$ -матрицы Z :

$$\begin{aligned} -\text{tr}(F_0 Z) &\rightarrow \max, \\ \text{tr}(F_i Z) = c_i, \quad i = 1, \dots, m, \\ Z &\geq 0. \end{aligned} \tag{1.3}$$

Матрица Z является допустимой для двойственной задачи (1.3), если $\text{tr}(F_i Z) = c_i$, $i = 1, \dots, m$ и $Z \geq 0$. Условие регулярности для двойственной задачи (1.3) имеет вид: существует $n \times n$ -матрица Z такая, что $\text{tr}(F_i Z) = c_i$, $i = 1, \dots, m$ и $Z > 0$ (т.е. последнее неравенство — строгое).

Пусть $p(x) = c'x$ и $d(Z) = -\text{tr}(F_0 Z)$ — целевые функции для прямой и двойственной задач соответственно. Рассмотрим функцию, линейно зависящую от x и Z :

$$\eta(x, Z) = p(x) - d(Z) = c'x + \text{tr}(F_0 Z).$$

Если вектор x допустим для прямой задачи, а матрица Z допустима для двойственной задачи, то

$$\eta(x, Z) = \text{tr}(F(x)Z) \geq 0, \tag{1.4}$$

т.е. $p(x) \geq d(Z)$. Функция $\eta(x, Z)$ называется величиной разрыва двойственности (duality gap).

Обозначим p^* — оптимальное значение целевой функции прямой задачи, а d^* — оптимальное значение двойственной задачи (в предположении, что оптимальные решения этих задач существуют). Для того, чтобы не вдаваться в технические подробности, сделаем довольно сильное предположение о том, что как для прямой, так и для двойственной задачи выполнены условия регулярности. Тогда обе оптимизационные задачи имеют решения, значения p^* и d^* конечны и $p^* = d^*$. (В этом случае говорят, что имеет место сильная двойственность.) Если оптимальные значения достигаются на допустимых точках x^* и Z^* , то $\eta(x^*, Z^*) = 0$, т.е. разрыв двойственности отсутствует. Из (1.4) следует, что $\text{tr}(F(x^*)Z^*) = 0$. Тогда неотрицательная определенность матриц $F(x^*)$ и Z^* влечет равенство $F(x^*)Z^* = 0$. Последнее соотношение называется условием дополняющей нежесткости.

Таким образом, решение прямой и двойственной задачи может быть заменено на решение смешанной задачи полуопределенного программирования (primal-dual problem)

$$\begin{aligned} c'x + \text{tr}(F_0Z) &\rightarrow \min, \\ F(x) &\geq 0, \quad Z \geq 0, \\ \text{tr}(F_iZ) &= c_i, \quad i = 1, \dots, m. \end{aligned} \tag{1.5}$$

В случае, если она успешно решается, оптимальное значение целевой функции должно быть равно нулю.

1.4. Численные методы исследования линейных матричных неравенств

Для численного исследования линейных матричных неравенств в настоящее время применяется несколько общих подходов. Наиболее эффективной считается группа методов, носящих название методов внутренней точки.

Метод внутренней точки (другие названия — метод внутренних штрафов, метод барьерных функций, метод последовательной безусловной оптимизации) хорошо известен в теории нелинейного программирования и упоминается во всех стандартных учебниках по этому разделу математики. Одно из первых его подробных обоснований было дано в 60-е годы в работах А. Фиакко и Г. П. Мак-Кормика (имеется русский перевод [24]). Опишем идею метода внутренней точки применительно к традиционной постановке задачи нелинейного программирования.

Рассмотрим задачу оптимизации с ограничениями

$$\begin{aligned} f(x) &\rightarrow \min, \\ f_i(x) &\leq 0, \quad i = 1, \dots, n, \end{aligned} \tag{1.6}$$

где $f(x)$, $f_i(x)$ — выпуклые скалярные функции, зависящие от векторов $x \in \mathbb{R}^m$. Предположим, что область строго допустимых векторов задачи (1.6), т.е. множество таких x , что $f_i(x) < 0$, $i = 1, \dots, n$, непуста. Функция $\phi(x)$ называется барьерной для области строго допустимых векторов задачи (1.6), если она обладает свойством: когда x стремится к границе этой области, оставаясь внутри нее, функция $\phi(x) \rightarrow +\infty$. Таким образом, функция $\phi(x)$ задает «барьер» для выхода точки x из области строгой допустимости. При этом вместо исходной задачи можно рассмотреть задачу оптимизации без ограничений

$$f(x) + r\phi(x) \rightarrow \min, \tag{1.7}$$

где r — некоторый положительный параметр. Исходя из некоторого начального строго допустимого вектора, для ее решения можно применить какой-либо из методов безусловной оптимизации, например метод Ньютона с регулировкой шага [16]. Допустим, что задача (1.7) разрешима и минимум достигается в строго допустимой точке $x(r)$. При достаточно слабых предположениях получаем, что $x(r)$ стремится к оптимальному решению исходной задачи (1.6) при $r \rightarrow +0$ (точные математические утверждения можно найти, например, в [4, 10, 16, 18]).

При замене задачи (1.6) на задачу (1.7) возникает следующая проблема. Для того, чтобы решение (1.7) достаточно точно приближало решение задачи (1.6), требуется малость параметра r . В то же время, при малых r методы решения задачи (1.7) плохо сходятся. (Содержательное обсуждение этого явления можно найти в [7] — оно связано с плохой обусловленностью матрицы Гессе целевой функции задачи (1.7).) Для разрешения этого противоречия строят последовательность r_k значений параметра r такую, что $r_k > r_{k+1}$ и $r_k \rightarrow +0$ при $k \rightarrow \infty$. Для каждого r_k находят оптимальную точку x_k , выбирая точку x_{k-1} в качестве начального приближения.

Начальную строго допустимую точку можно выбрать, например, решая вспомогательную оптимационную задачу относительно вектора x и числа M :

$$\begin{aligned} M &\rightarrow \min, \\ f_i(x) &\leq M, \quad i = 1, \dots, n. \end{aligned}$$

В качестве барьерной функции часто выбирают так называемый логарифмический барьер

$$\phi(x) = - \sum_{i=1}^n \ln(-f_i(x)).$$

В случае гладких и выпуклых функций $f_i(x)$ он также обладает свойством гладкости и выпуклости. При рассмотрении логарифмического барьера точки оптимума $x(r)$ иногда называют центрами, а параметрически заданную при $r > 0$ кривую $x(r)$ — центральным путем (central path) [37].

Приведенные выше рассуждения были перенесены на линейные матричные неравенства. При этом особо важную роль сыграли работы сотрудников ЦЭМИ РАН А. С. Немировского и Ю. Е. Нестерова, которые в 80-е годы подробно разработали методику применения методов внутренней точки к ряду задач, включая задачи полуопределенного программирования. К числу основных результатов Немировского и Нестерова относится получение простых условий, обеспечивающих алгоритмам внутренней точки так называемую «полиномиальную сложность». Итоги их исследования подведены в монографии [54]. За ней последовал ряд книг [38, 60, 63, 67, 69] (впрочем, мало доступных в России) и большое количество статей. Интересные материалы на эту тему можно найти на сайте возглавляемого А. С. Немировским Центра оптимизации MINERVA <http://iew3.technion.ac.il/Labs/Opt/>.

Методы внутренней точки для задач полуопределенного программирования имеют несколько вариантов. Наиболее известные из них — метод аналитических центров, метод одновременного решения прямой и двойственной задачи, проективный метод.

Большинство вариантов метода внутренней точки для задач полуопределенного программирования используют логарифмический барьер. Рассмотрим симметричную матричную функцию $F(x)$ размера $n \times n$, аффинно зависящую от x , т.е.

$$F(x) = F_0 + \sum_{i=1}^m x_i F_i.$$

Будем предполагать, что матрицы F_1, \dots, F_m линейно независимы. Допустим, что область

$$\Omega = \{x \in \mathbb{R}^m \mid F(x) > 0\}$$

непуста и ограничена. Построим для нее барьерную функцию по формуле:

$$\phi(x) = \ln \det F(x)^{-1}, \quad x \in \Omega.$$

Очевидно, функция $\phi(x) = -\ln \det F(x)$ и $\phi(x) \rightarrow +\infty$, если x изнутри приближается к границе области Ω , для которой выполнено $\det F(x) = 0$.

Градиент функции $\nabla \phi(x)$ и ее матрица Гессе (матрица, составленная из вторых производных) $\nabla^2 \phi(x)$ могут быть вычислены по формулам

$$[\nabla \phi(x)]_i = -\operatorname{tr} F(x)^{-1} F_i, \quad [\nabla^2 \phi(x)]_{ij} = -\operatorname{tr} F(x)^{-1} F_i F(x)^{-1} F_j.$$

Несмотря на довольно простой вид функции $\phi(x)$, вывод этих формул не слишком очевиден — подробные выкладки могут быть найдены, например, в [36]. Там же показано, что из формулы для матрицы Гессе и линейной независимости матриц F_1, \dots, F_m следует положительная определенность матрицы Гессе на всем множестве Ω . Отсюда функция $\phi(x)$ является строго выпуклой на Ω и поэтому имеет на Ω точку минимума x^* , причем единственную. Эта точка называется аналитическим центром неравенства $F(x) > 0$. (Подчеркнем, что понятие аналитического центра связано именно с неравенством, а не с множеством Ω .) Для нахождения аналитического центра может быть использован один из методов безусловной оптимизации, например, метод Ньютона. Существуют различные формулы для регулировки длины шага в методе Ньютона (наиболее известные из них были предложены Немировским и Нестеровым).

Рассмотрим теперь оптимизационную задачу

$$\begin{aligned} c'x &\rightarrow \min, \\ F(x) &> 0. \end{aligned}$$

По аналогии с рассмотренной выше задачей нелинейного программирования ее можно заменить на задачу безусловной оптимизации

$$c'x + \mu \ln \det F(x)^{-1} \rightarrow \min$$

на множестве Ω . Здесь μ — положительный параметр, который, убывая, стремится к нулю.

Существует множество других вариантов построения барьеров и вспомогательных оптимизационных задач. В частности, можно рассмотреть задачу [35]

$$\ln(\lambda - c'x)^{-1} + \ln \det F(x)^{-1} \rightarrow \min$$

при x таких, что $x \in \Omega$ и $\lambda > c'x$. Здесь λ — убывающий положительный параметр.

Задачу исследования разрешимости неравенства $F(x) > 0$ (и одновременно поиска начального приближения для других оптимизационных задач) можно решить с помощью вспомогательной задачи

$$\begin{aligned} t &\rightarrow \min, \\ F(x) &> -tI_n, \end{aligned}$$

где t — скалярная переменная. Неравенство разрешимо, если на каком-либо шаге получаем значение $t < 0$. (Таким образом, итерационный процесс можно прекратить даже если минимальное значение t еще не достигнуто.)

Начиная с середины 90-х годов, наиболее популярными являются алгоритмы внутренней точки, основанные на одновременном решении прямой и двойственной задачи (primal-dual methods). Именно их использует большинство современных программ-решателей. Например, исходя из задачи (1.5), можно определить новую задачу [64]

$$\begin{aligned} \ln \det F(x)^{-1} + \ln \det Z^{-1} &\rightarrow \min, \\ F(x) &> 0, \quad Z > 0, \\ \text{tr}(F_i Z) &= c_i, \quad i = 1, \dots, m, \\ c'x + \text{tr}(F_0 Z) &= \eta. \end{aligned}$$

Здесь η — убывающий положительный параметр, $\eta \rightarrow +0$, который представляет собой величину разрыва двойственности (duality gap). При фиксированном значении η можно найти решение прямой и двойственной задачи $x(\eta)$, $Z(\eta)$. Процесс вычисления прекращается, когда разрыв двойственности η становится достаточно малым.

Имеются и иные подходы к решению рассматриваемых задач: методы отсекающей плоскости [34] (по-видимому, наиболее известный метод этой группы — метод эллипсоидов [13, 35, 61]), а также методы, использующие субградиент. С точки зрения скорости сходимости они уступают в эффективности методам внутренней точки. Тем не менее, в [45, 46] указаны классы задач, для которых использование метода отсекающей плоскости может дать определенный выигрыш.

1.5. Матричные переменные

В прикладных задачах часто приходится иметь дело не со скалярными, а с матричными переменными. Примером может служить задача, связанная с неравенством Ляпунова: найти симметричную положительно

определенную матрицу H такую, что

$$HA + A'H < -Q.$$

Здесь A и $Q = Q'$ — заданные матричные коэффициенты, H — неизвестная матричная переменная. Очевидно, сформулированная выше задача может быть сведена к задаче разрешимости (FEASP), но это сведение потребует некоторых усилий.

Для того, чтобы непосредственно применить описанные выше численные алгоритмы, требуется предварительно перейти от матричной переменной к скалярным переменным. Число скалярных переменных, которые должны быть сопоставлены матричной переменной, зависит от предположений относительно структуры матрицы. Например, произвольной матрице M размера 2×2 отвечает четыре скалярных переменных:

$$M = \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}.$$

Если дополнительно предположить, что матрица M симметричная, то ей отвечают три скалярные переменные:

$$M = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \end{bmatrix}.$$

Если же матрица M диагональная, то ей отвечают две скалярные переменные

$$M = \begin{bmatrix} x_1 & 0 \\ 0 & x_2 \end{bmatrix}.$$

Матричные неравенства могут быть переписаны в виде $F(x) > 0$ или $F(x) \geqslant 0$.

В качестве примера рассмотрим упомянутое выше неравенство Ляпунова:

$$HA + A'H < -Q, \quad H > 0,$$

где H — симметричная матричная переменная размера 2×2 , A и Q — матричные коэффициенты такого же размера, $Q' = Q$. Представим H и A в виде

$$H = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

Тогда исходная система неравенств может быть переписана в виде

$$\begin{aligned} -Q + x_1 \begin{bmatrix} -2a_{11} & -a_{12} \\ -a_{12} & 0 \end{bmatrix} + x_2 \begin{bmatrix} -2a_{21} & -a_{11} - a_{22} \\ -a_{11} - a_{22} & -2a_{12} \end{bmatrix} + \\ + x_3 \begin{bmatrix} 0 & -a_{21} \\ -a_{21} & -2a_{22} \end{bmatrix} > 0, \end{aligned}$$

$$x_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} > 0.$$

Пакеты прикладных программ, которые будут рассмотрены в следующих главах, позволяют формулировать исходную задачу в терминах матричных переменных. Функции этих пакетов преобразуют матричные переменные в скалярные и пересчитывают коэффициенты неравенств автоматически, без дополнительных усилий со стороны пользователя.

Во многих случаях исходная задача сформулирована не в терминах линейных матричных неравенств, но может быть сведена к ним с помощью специальных приемов. Рассмотрим два наиболее важных таких приема: формулы Шура и S -процедуру.

1.6. Формулы Шура

Важным инструментом при сведении задачи к системе матричных неравенств является использование формул Шура для блочных матриц [5].

Теорема 1.1. Пусть задана симметричная блочная матрица

$$M = \begin{bmatrix} Q & P \\ P' & R \end{bmatrix}.$$

Тогда условие $M > 0$ эквивалентно каждой из двух систем неравенств:

$$Q - PR^{-1}P' > 0, \quad R > 0$$

или

$$R - P'Q^{-1}P > 0, \quad Q > 0.$$

Пример 1. Пусть симметричная положительно определенная матрица H задает эллипсоид

$$\Omega = \{x \in \mathbb{R}^n : x'Hx \leqslant 1\}.$$

Требуется, чтобы этот эллипсоид содержался в полосе пространства \mathbb{R}^n , определяемой с помощью вектора c и числа $\Delta > 0$:

$$\Pi = \{x \in \mathbb{R}^n : |c'x| < \Delta\}.$$

Нетрудно убедиться, что

$$\sup_{x \in \Omega} |c'x| = \sqrt{c'H^{-1}c}.$$

Поэтому условие $\Omega \subset \Pi$ эквивалентно условию:

$$c'H^{-1}c < \Delta^2.$$

Согласно формулам Шура, его можно заменить на неравенство, линейное относительно H :

$$\begin{bmatrix} H & c \\ c' & \Delta^2 \end{bmatrix} > 0.$$

Пример 2. Оценим максимального сингулярное число прямоугольной матрицы. Если $A(x)$ — прямоугольная $m \times n$ матрица, а t — скаляр, то неравенство $A'(x)A(x) < tI_n$ эквивалентно неравенству

$$\begin{bmatrix} I_m & A(x) \\ A'(x) & tI_n \end{bmatrix} > 0.$$

Пример 3 (M. de Oliveira). Рассмотрим задачу с квадратичной функцией цели:

$$\begin{aligned} x'Q'Qx + c'x &\rightarrow \min, \\ F(x) &\leq 0, \end{aligned}$$

где c — вектор, Q — прямоугольная матрица. Введем дополнительную скалярную переменную γ и заменим исходную задачу на следующую:

$$\begin{aligned} \gamma &\rightarrow \min, \\ \gamma &\geq x'Q'Qx + c'x, \quad F(x) \leq 0. \end{aligned}$$

Воспользуемся формулами Шура и получим эквивалентную задачу полуопределенного программирования:

$$\begin{aligned} \gamma &\rightarrow \min, \\ \begin{bmatrix} \gamma - c'x & x'Q' \\ Qx & I \end{bmatrix} &\geq 0, \quad F(x) \leq 0. \end{aligned}$$

Пример 4. Квадратичное матричное неравенство (алгебраическое неравенство Риккати)

$$R + HP + P'H - HQQ'H \geq 0,$$

где H — симметричная матричная переменная, P, Q, R — матричные коэффициенты, может быть сведено к эквивалентному линейному матричному неравенству

$$\begin{bmatrix} R + HP + P'H & HQ \\ Q'H & I \end{bmatrix} \geq 0.$$

Пример 5. Рассмотрим задачу минимизации линейной формы на множестве квадратичных конусов (second-order cone programming, SOCP):

$$\begin{aligned} f'x &\rightarrow \min, \\ \|A_i x + b_i\| &\leq c'_i x + d_i, \quad i = 1, \dots, N. \end{aligned}$$

Здесь x — n -мерный вектор, составленный из скалярных переменных, f — n -мерный вектор-коэффициент, A_i — прямоугольные матрицы, b_i и c_i — вектора, подходящей размерности. Под $\|\cdot\|$ понимается евклидова норма вектора. SOCP имеет многочисленные приложения в механике и технике (см., например, [32, 50]). Используя формулы Шура, эту задачу можно свести к стандартной задаче полуопределенного программирования [50]:

$$\begin{aligned} f'x &\rightarrow \min, \\ \begin{bmatrix} (c'_i x + d_i)I & A_i x + b_i \\ (A_i x + b_i)' & c'_i x + d_i \end{bmatrix} &\geq 0, \quad i = 1, \dots, N. \end{aligned}$$

Тем не менее, выполнять такое сведение не рекомендуется. Дело в том, что для задачи SOCP существуют специализированные эффективные алгоритмы решения, которые реализованы в виде программных пакетов. Более подробное ее рассмотрение остается за рамками этой книги.

Прием, связанный с использованием формул Шура, оказывается полезен и во многих других ситуациях.

1.7. Некоторые примеры

Пример 1. Рассмотрим оптимизационную задачу

$$\begin{aligned} g'H^{-1}g &\rightarrow \min, \\ \begin{bmatrix} HA + A'H & Hb \\ b'H & 0 \end{bmatrix} &\leq \begin{bmatrix} \mu_1 \mu_2 cc' & -\frac{1}{2}(\mu_1 + \mu_2)c \\ -\frac{1}{2}(\mu_1 + \mu_2)c' & 1 \end{bmatrix}. \end{aligned}$$

Здесь A , b , c , g , μ_1 , μ_2 — известные коэффициенты, причем A — $n \times n$ -матрица, b , c и g — n -векторы, μ_1 , μ_2 — скаляры. Симметричная $n \times n$ -матрица H является матричной переменной. Покажем, как эта задача может быть сведена к задаче полуопределенного программирования.

Если дополнительно предположить, что матрица H положительно определена, то вместо целевого условия

$$g'H^{-1}g \rightarrow \min$$

можно рассмотреть задачу

$$t \rightarrow \min, \quad g' H^{-1} g \leq t,$$

которая сводится к задаче

$$t \rightarrow \min, \quad \begin{bmatrix} t & g' \\ g & H \end{bmatrix} \geq 0.$$

В случае, когда положительная определенность H не предполагается, сведение к стандартной оптимизационной задаче требует несколько больше усилий [25]. Преобразуем неравенство, задающее ограничение исходной задачи: умножим его справа на матрицу

$$\begin{bmatrix} I_n & 0 \\ \mu_1 c' & 1 \end{bmatrix},$$

а слева — на транспонированную матрицу. Получим

$$\begin{bmatrix} H(A + \mu_1 bc') + (A' + \mu_1 cb')H & Hb \\ b'H & 0 \end{bmatrix} \leq \begin{bmatrix} 0 & -\frac{1}{2}(\mu_2 - \mu_1)c \\ -\frac{1}{2}(\mu_2 - \mu_1)c' & 1 \end{bmatrix}.$$

Умножим это неравенство слева и справа на симметричную матрицу

$$\begin{bmatrix} H^{-1} & 0 \\ 0 & 1 \end{bmatrix}.$$

Вместо матричной переменной H рассмотрим матричную переменную $K = H^{-1}$. В результате получим стандартную задачу

$$g' K g \rightarrow \min,$$

$$\begin{bmatrix} K(A' + \mu_1 cb') + (A + \mu_1 bc')K & -\frac{1}{2}(\mu_2 - \mu_1)Kc \\ -\frac{1}{2}(\mu_2 - \mu_1)c'K & 0 \end{bmatrix} \leq \begin{bmatrix} 0 & b \\ b' & 1 \end{bmatrix}.$$

Пример 2. Рассмотрим задачу нахождения эллипсоида наименьшего объема, содержащего заданный набор векторов $s_i \in \mathbb{R}^n$, $i = 1, \dots, m$ [35].

Опишем эллипсоид соотношением

$$\Omega = \{x \mid x' H x \leq 1\},$$

где H — некоторая положительно определенная $n \times n$ -матрица. Тогда условие $s_i \in \Omega$ эквивалентно неравенству $s_i' H s_i \leq 1$. Объем эллипсоида вычисляется по формуле

$$\text{Vol}(\Omega) = V_0 \sqrt{\det H^{-1}},$$

где V_0 — объем шара единичного радиуса в \mathbb{R}^n . Таким образом, получаем оптимизационную задачу

$$\ln \det H^{-1} \rightarrow \min, \quad s_i' H s_i \leq 1, \quad i = 1, \dots, m.$$

Целевое условие можно переписать в виде

$$\ln \det H \rightarrow \max.$$

Пример 3. Рассмотрим близкую задачу: найти эллипсоид Ω_0 наибольшего объема, который содержится в пересечении гиперплоскостей

$$\Omega_0 \subset \{x \mid |c_i' x| \leq 1, \quad i = 1, \dots, m\},$$

здесь c_i — n -мерные векторы-столбцы. Представим эллипсоид в виде

$$\Omega_0 = \{x \mid x' H^{-1} x \leq 1\},$$

где H — некоторая положительно определенная матрица. Так как

$$\max_{x \in \Omega_0} |c_i' x| = \sqrt{c_i' H c_i}$$

и объем эллипсоида пропорционален величине $(\det H)^{1/2}$, приходим к оптимизационной задаче [35]:

$$\ln \det H \rightarrow \max, \quad c_i' H c_i \leq 1, \quad i = 1, \dots, m.$$

1.8. S-процедура

S-процедура — это специальный технический прием преобразования квадратичных или эрмитовых форм. По-видимому, первые варианты *S*-процедуры появились в работах А.И. Лурье [11]. В статье [27] было дано строгое математическое обоснование так называемой «неупербности» *S*-процедуры для случая одного квадратичного ограничения.

Теорема 1.2. Пусть $F(u), G(u)$ — две квадратичные формы, определенные на \mathbb{R}^n , причем существует такой вектор u_0 , что $G(u_0) > 0$. Следующие два условия эквивалентны:

1. $F(u) \geq 0$ на множестве $\{u \mid G(u) \geq 0\}$.
2. Существует число $\tau \geq 0$ такое, что $F(u) - \tau G(u) \geq 0$ при всех u .

Аналогичные утверждения справедливы для пары эрмитовых форм (т.е. в комплексном пространстве). Сформулированную выше теорему обычно называют теоремой о неущербности S -процедуры, она часто используется при работе с линейными матричными неравенствами.

К сожалению, в случае, когда вместо одного квадратичного ограничения $G(u) \geq 0$ рассматривается несколько квадратичных ограничений $G_1(u) \geq 0, \dots, G_m(u) \geq 0$, теорема о неущербности S -процедуры не верна. Можно лишь утверждать, что из существования неотрицательных чисел τ_1, \dots, τ_m таких, что

$$F(u) - \sum_{i=1}^m \tau_i G_i(u) \geq 0 \quad \text{при всех } u,$$

следует, что $F(u) \geq 0$ на множестве $\{u \mid G_1(u) \geq 0, \dots, G_m(u) \geq 0\}$.

Обозначим через \hat{F} и \hat{G} матрицы квадратичных форм $F(u)$ и $G(u)$ соответственно. Пусть эти матрицы аффинно зависят от некоторых неизвестных параметров, сведенных в вектор x : $\hat{F} = \hat{F}(x)$, $\hat{G} = \hat{G}(x)$. Рассмотрим множество векторов x , для которых выполнено условие 1 теоремы об S -процедуре. Из теоремы следует, что это множество может быть описано линейным матричным неравенством $\hat{F}(x) - \tau \hat{G}(x) \geq 0$.

1.9. Решатели и интерфейсные пакеты

В настоящее время в Интернете можно найти большое количество разнообразных пакетов программ, предназначенных для решения линейных матричных неравенств и связанных с ними оптимизационных задач. Большинство этих пакетов ориентированы на использование в системе MATLAB. В основном они разрабатывались (и продолжают разрабатываться) как инициативные проекты, а их использование является бесплатным. В конце этого раздела приведена таблица со списком интернет-адресов сайтов наиболее известных пакетов (по состоянию на лето 2004 года).

Программы исследования линейных матричных неравенств можно разбить на два класса. Программы первого класса в литературе носят название решающих программ (solvers). Они предназначены для исследования линейных матричных неравенств, заданных с помощью скалярных переменных. В этой книге мы не ставим своей целью подробный анализ программ-решателей, которые очень разнообразны и зачастую имеют тонкие различия в алгоритмах.

Наиболее старый из пакетов этой группы носит название SP (Semidefinite Programming package), разработчики Lieven Vandenberghe, Univer-

sity of California, Los Angeles и Stephen Boyd, Stanford University (первая версия была выпущена в 1994 г.). В 2004 г. разработчики объявили SP устаревшим и прекратили его дальнейшее распространение.² В настоящее время имеется большое количество других программ-решателей: SDPA, SDPHA, SDPack, SDPT3, CSDP, CUTSDP, DSDP, COPL_DSDP, SeDuMi, MAXDET, SOCP, PENSDP. (Список неполный, т.к. число разработок в этой области очень велико.) Практически все они используют различные варианты алгоритмов внутренней точки. Иногда применяется также метод отсекающей гиперплоскости (решатель CUTSDP). Заметим, что разработчики решателей обычно не предоставляют свободный доступ к исходным кодам своих программ. Сообщается лишь о типе применяемого численного метода, а детали реализации скрыты в библиотеках динамической компоновки (DLL).

В настоящее время преобладают решатели, использующие одновременное решение прямой и двойственной задачи (primal-dual methods), причем наиболее распространена модификация этого метода, принадлежащая Нестерову и Тодду [53]. По-видимому, самый популярный решатель этого типа — SeDuMi (Self Dual Minimization), разработчик Jos Sturm, Universiteit van Tilburg, Нидерланды (последняя версия выпущена в 2001 г.). SeDuMi обладает высокой надежностью и является одним из самых быстрых. Кроме того, он непосредственно (без сведения к задаче полуопределенного программирования) решает задачу минимизации линейной целевой функции при ограничениях типа квадратичных коэффициентов (SOCP). К большому сожалению, развитие этого замечательного проекта было прервано безвременной смертью его автора, Йоса Штурма, в декабре 2003 г. Среди совсем новых решателей отметим решатель PENSDP, созданный в рамках коммерческого проекта PENOPT и реализующий новый алгоритм построения барьера в методе внутренней точки (penalty-barrier multiplier method) [33].

Однако для приложений к задачам теории управления использование скалярных переменных неудобно, более предпочтительной является работа с матричными переменными. Конечно, любая задача в матричной постановке может быть переформулирована в виде, подходящем для входа решающей программы, однако этот процесс может быть весьма трудоемок. Для автоматизации преобразования матричных переменных в скалярные (с соответствующим пересчетом коэффициентов) служат программы второго класса — интерфейсные. В англоязычной литературе для них часто используют термины front-end (внешний интерфейс)

²Создатели SP рекомендуют для использования решатели SeDuMi, SDPT3, SDPA, CSDP и интерфейсный пакет YALMIP.

или parser (синтаксический анализатор). Зачастую один и тот же пакет предоставляет интерфейс для работы не с одним, а с несколькими решателями. Подробные сведения об интерфейсных пакетах будут приведены в следующем разделе.

Существует ряд прикладных пакетов, для которых линейные матричные неравенства являются не целью, а только аппаратом исследования. К ним, в частности, относится подпакет Control Applications пакета LMI Control Toolbox системы MATLAB.

Отметим также интересный пакет IQC β (Integral Quadratic Constraints), предназначенный для работы в MATLAB и SIMULINK [52], разработчики A. Megretski (MIT), C. Kao и A. Rantzer (Lunds universitet, Швеция), U. Jönsson (Kungliga Tekniska högskolan, Stockholm). Он использует LMILab и может служить хорошей иллюстрацией применения этого пакета при решении задач теории управления, которые формулируются в терминах интегральных квадратичных ограничений (IQC). Для многих задач теории управления использование IQC является альтернативой использования LMI, и сейчас численные аспекты этого подхода интенсивно изучаются. Например, в статье [45] для исследования задач разрешимости и оптимизации в форме IQC предложены новые алгоритмы отсекающей плоскости, которые, по утверждению авторов, эффективнее ставших традиционными методов внутренней точки.

Существуют и другие программные разработки, решающие близкие прикладные задачи. Например, пакеты связанные с исследованием квадратичных матричных неравенств или неравенств, содержащих полиномиальные матрицы. Наиболее известный продукт такого рода — пакет PolyX (Polynomial Toolbox) для системы MATLAB [57]. Отметим также пакеты глобальной полиномиальной оптимизации GloptiPoly [43] и SOSTOOLS [58] (Sums of Squares optimization tools).

И наконец, в главе этой книги, посвященной пакету YALMIP, можно найти сведения о современных пакетах, предназначенных для исследования билинейных матричных неравенств (bilinear matrix inequalities, BMI).

Данная область теории оптимизации развивается необычайно бурно, постоянно возникают новые программы и обновляются старые. Следить за новинками без помощи Интернета практически невозможно. В приложении 2 приведены адреса, по которым можно найти некоторые из перечисленных пакетов. К сожалению, ссылки часто меняются, поэтому для их уточнения рекомендуется использовать поисковые системы.

1.10. Обзор интерфейсных пакетов исследования LMI

Рассмотрение интерфейсных пакетов — основное содержание этой книги. Интерфейсные пакеты позволяют формулировать задачу в удобной форме, с помощью матричных коэффициентов и матричных неизвестных. Они преобразуют высокоуровневый код к такому виду, чтобы он мог быть воспринят решателем.

Можно выделить следующие типы интерфейсных пакетов.

1. Интерфейс рассчитан на работу с одним определенным решателем. Часто такие интерфейсы объединены с решателем в едином пакете.

2. Интерфейс рассчитан на работу с несколькими решателями.

3. Интерфейс является надстройкой над другим интерфейсом, обычно широко распространенным, с целью расширения его функциональности.

Как будет видно из дальнейшего описания, такое деление весьма условно, так как некоторые пакеты занимают пограничное положение. Разберем имеющиеся пакеты каждой из этих трех групп. Отметим, что приводимые оценки достоинств и недостатков пакетов основываются на нашем личном опыте и отчасти носят субъективный характер.

1.10.1. Интерфейсы для работы с одним решателем

LMILAB

Пакет LMILab — наиболее известный из пакетов для исследования линейных матричных неравенств. Он входит в стандартный дистрибутив системы MATLAB в составе более крупного пакета LMI Control Toolbox. Разработчики Pascal Gahinet и Mahmoud Chilali (The Mathworks, Inc.), Arkadi Nemirovski (Technion, Израиль), Allan Laub (University of California, Santa Barbara). Последняя версия пакета относится к 2002 г. По сути он представляет собой сложный комплекс программ, включающий в себя как несколько программ-решателей, так и интерфейсные программы различного назначения. Основной встроенный решатель реализует проективный алгоритм Немировского.

LMILab предоставляет удобную возможность с помощью графических окон интерактивно вводить данные о задаче оптимизации. Так же интерактивно (правда, в текстовом режиме) можно получать информацию о задаче. Пакет содержит широкий набор вспомогательных функций (получение подробной информации о задаче, взаимные преобразования матричных и скалярных переменных). Будучи стандартным пакетом MATLAB, он хорошо отлажен и снабжен подробной документацией.

В то же время LMILab мало эффективен при работе с задачами большей размерности и с разреженными матрицами. Существенный недостаток пакета состоит в том, что он работает только с ограничениями типа строгих неравенств; ограничения типа равенств и нестрогие неравенства не допускаются. Пакет не допускает непосредственной работы с комплексными числами — требуется вручную выделять вещественную и мнимую части.

Говоря в целом, это удобный пакет для задач с небольшим числом переменных, но он имеет неприятные ограничения.

LMITOOL

Пакет LMITOOL существует в двух вариантах — для систем Scilab и MATLAB. Остановимся подробнее на варианте для системы Scilab. Так же, как LMILab входит в дистрибутив MATLAB, LMITOOL входит в дистрибутив Scilab. Разработчики Ramine Nikoukhah и Francois Delebecque (INRIA), Laurent El Ghaoui (ENSTA, ныне работает в Berkeley University). Пакет интегрирован с решателем SP, который также включен в Scilab. Этот пакет реализует один из первых алгоритмов одновременного решения прямой и двойственной задачи (primal-dual method).

В отличие от LMILab, пакет LMITOOL допускает рассмотрение ограничений типа равенств. Наряду с программным способом определения оптимизационной задачи, пакет имеет удобный графический интерфейс пользователя (GUI).

В то же время, в пакете доступно меньшее, чем в LMILab, количество оптимизационных задач. В LMILab можно решать четыре вида задач — FEASP, SDP, GEVP и MAXDET, в LMITOOL — только первые два вида. Решатель SP, который используется в Scilab, старый и не слишком эффективный.

С вариантом пакета, предназначенным пользователям MATLAB, произошли некоторые изменения. Версия 2.1 (1999 г.) предоставляла выбор из трех решателей — SP, SDPack и SDPHA, однако с конца 2003 года доступ в Интернете к этому пакету прекращен. Версия 2.2 (2001 г.) допускает работу с единственным решателем — SeDuMi.

SEDUMI INTERFACE

SeDuMi Interface — чисто интерфейсный пакет (разработчики Dimitri Peaucelle, Didier Henrion, Yann Labit, Krysten Taitz, все из LAAS-CNRS, Toulouse). Как ясно из названия, он ориентирован на работу с решающей программой SeDuMi в среде MATLAB. Как уже отмечалось, SeDuMi,

по-видимому, самый популярный в настоящее время решатель. Он реализует один из вариантов алгоритма одновременного решения прямой и двойственной задачи. Обе программы, SeDuMi и SeDuMi Interface, бесплатные, они допускают работу в различных версиях Windows и UNIX.

В отличие от LMILab и LMITOOL, которые реализованы как библиотеки функций, SeDuMi Interface имеет объектную организацию. В нем определен единственный класс, и большинство функций пакета являются его методами.

В пакете предусмотрена возможность решения задач с ограничениями типа равенств. При этом сам пакет выполняет автоматическое исключение зависимых переменных и тем самым уменьшает размерность задачи. Кроме того, SeDuMi Interface дает возможность работы с комплексными матрицами (как эрмитовыми так и антиэрмитовыми), что является существенным преимуществом перед другими пакетами. В пакете предусмотрена возможность использования дополнительных синтаксических конструкций: оператора следа при задании целевой функции, а также прямого произведения Кронекера при задании ограничений.

Оптимизационная задача в SeDuMi Interface создается чисто программно, какие-либо средства ее интерактивного определения отсутствуют. Набор сервисных функций пакета довольно ограничен, например, нельзя проверить структуру введенного ограничения. В этом отношении он значительно уступает пакету LMILab. В 2003 г. авторы объявили о прекращении дальнейшей модификации этого продукта и об открытии нового проекта под названием LiMaCOS.

SDPSOL

Пакет SDPSOL (разработчики Shao-Po Wu и Stephen Boyd, Stanford University) — один из первых интерфейсных пакетов. В качестве решателя используется пакет SP. Он может работать самостоятельно (только в операционной системе UNIX) или в рамках системы MATLAB [68], при этом решатель SP требует отдельной установки. Для самостоятельного использования SDPSOL содержит встроенный язык описания задачи, по сути представляющий собой подмножество MATLAB.

Класс задач, решаемых с помощью SDPSOL, достаточно широк: допускаются ограничения типа равенств, целевая функция может содержать определители. Тем не менее, неравенства могут быть только строгие. В пакете почти полностью отсутствуют сервисные функции.

По сути этот пакет представляет собой экспериментальную разработку. Он рекомендован авторами проекта только для задач небольшой размерности и сейчас интересен в основном в историческом плане, как один

из первых (если не самый первый) интерфейсный пакет (SDPSOL выпущен в 1996 году, первая версия решателя SP — в 1994 году). Для своего времени он, безусловно, являлся крупным достижением.

LMISOL

Пакет LMISol (разработчики Mauricio C. de Oliveira, Daniela P. de Farias и José C. Geromel, Universidade Estadual de Campinas, Бразилия, 1997 г.) также представляет собой самостоятельный программный продукт, разработанный в среде программирования C++. Пакет позволяет решать задачи выпуклого программирования с помощью как скалярных, так и матричных переменных, допускаются ограничения типа равенств и неравенств. В качестве решателя используется проективный метод Немировского (как и в LMILab). LMISol использует собственный язык программирования, напоминающий язык MATLAB. В настоящее время имеются версии пакета для операционных систем Linux и Solaris [39]. К сожалению, версия для Windows у этого продукта отсутствует. Пакет распространяется бесплатно (предварительно нужно зарегистрироваться) и снабжен подробной документацией.

1.10.2. Интерфейсы для работы с несколькими решателями

YALMIP

Интерфейсный пакет YALMIP (сокращение фразы «Yet Another LMI Parser» — «еще один анализатор линейных матричных неравенств») [51], разработчик Johan Löfberg, Linköpings universitet, Швеция, ныне работает в Eidgenössische Technische Hochschule (ETH), Zürich, Швейцария. Последняя версия продукта (YALMIP 3) выпущена в январе 2004 года. Этот пакет сейчас интенсивно развивается, появляются новые модификации. Пакет прошел тестирование в операционных системах Windows XP, Solaris и Linux.

Основная особенность этого пакета — очень большой выбор решающих программ (более двадцати). К ним относятся не только решатели, связанные с полуопределенным программированием, но и решатели, относящиеся к линейному, целочисленному, квадратичному программированию, некоторым полиномиальным задачам. Могут быть использованы как бесплатные, так и коммерческие пакеты. В частности, поддерживаются решатели SeDuMi, DSDP, SDPT3, CSDP, SDPA, MAXDET, PENSDP. Таким образом, пользователь имеет возможность выбрать себе решатель, учитывая особенности задачи (вид целевой функции, большое или малое число переменных, наличие разреженных матриц).

Пакет использует объектные типы данных. Широко применяется такое средство объектно-ориентированного программирования как перегрузка операций. Это позволяет описывать задачу в виде, напоминающем естественные математические формулы. Поддерживаются ограничения типа равенств, строгие и нестрогие неравенства, комплексные переменные. Интерактивный режим работы в пакете отсутствует.

В настоящее время YALMIP находится в стадии активной модификации, постоянно вносятся мелкие изменения, выявляются и исправляются ошибки, что создает определенные неудобства.³ В целом можно утверждать, что этот пакет наиболее современный и предоставляет наибольшие возможности для работы. Его можно рекомендовать пользователю, не склонному к консерватизму и готовому идти на разумный риск.

LiMACOS

Пакет LiMaCOS (Linear Matrix Constraints) пока еще только анонсирован. Он будет представлять собой дальнейшее развитие пакета SeDuMi Interface, коллектив разработчиков в основном тот же. Заявлены особенности пакета: он обеспечивает работу с несколькими решателями, включая SeDuMi, SDPack и SDPT3, использует графический интерфейс для создания задачи. В 2004 г. в связи с выходом новой версии пакета YALMIP, который во многом дублирует возможности LiMaCOS, работа над проектом была приостановлена. В сложившейся обстановке авторы LiMaCOS рекомендуют для использования YALMIP. Тем не менее, они не исключают возможность в дальнейшем вернуться к этой разработке.

1.10.3. Интерфейсы — надстройки над другими интерфейсами

LMILAB TRANSLATOR

Пакет LMILab Translator (разработчик Pete Seiler, Berkeley University, 2001 г.) можно отнести к пакетам как второй, так и третьей группы. С одной стороны, он предоставляет интерфейс для целого ряда решателей: CSDP, SeDuMi, SDPA, SDPHA, SDPack, SDPT3, SP. С другой стороны, он является надстройкой над стандартным пакетом LMILab.

Использование LMILab Translator предполагает, что исходная задача формируется средствами LMILab. В качестве альтернативы для функции `mincx`, входящей в состав LMILab, LMILab Translator предоставляет новую функцию `minsdp`, которая позволяет вызывать решатели других

³Как остроумно пишет сам Ю. Леффберг «... Если из-за ошибки в YALMIP ваш космический спутник потерпит катастрофу или защита вашей докторской диссертации провалится, то это ваша проблема».

пакетов. Анализ результата и модификация задачи также выполняются средствами LMILab. Разработчик пакета особо рекомендует его при работе с разреженными матрицами, когда LMILab мало эффективен.

xLMI

Небольшой пакет xLMI (разработчик Wes Thompson, University of Illinois at Urbana-Champaign) предназначен для тех пользователей, которые привыкли к стандартному пакету LMILab, но хотели бы воспользоваться решателем SeDuMi. Он реализует дополнительный интерфейс с пакетом SeDuMi Interface. (Таким образом, для его работы требуется одновременная установка трех пакетов — xLMI, SeDuMi и SeDuMi Interface.) Синтаксис функций пакета xLMI почти совпадает с синтаксисом LMILab (имена функций снабжаются дополнительным префиксом «x»).

KYPD

Интересным дополнением к YALMIP является небольшой (состоящий всего из двух функций) специализированный пакет KYPD, разработанный аспирантом Рагнаром Валлином (Ragnar Wallin, Linköpings universitet). Он предназначен для решения задачи полуопределенного программирования, связанной с леммой Калмана—Якубовича—Попова (в отечественной литературе она чаще называется частотной теоремой или леммой Якубовича—Калмана). При этом в качестве решателя используется SeDuMi.

В заключение высажем общее соображение — идеальных универсальных решателей и интерфейсов не существует. Если вы предъявляете особо жесткие требования к надежности результатов, проверьте расчеты в нескольких различных пакетах.

1.11. История метода линейных матричных неравенств в теории управления

Возникновение метода линейных матричных неравенств обычно относят к 1890 году, когда была впервые опубликована известная диссертация А. М. Ляпунова по теории устойчивости движения [12]. В рамках второго (прямого) метода Ляпунова в ней были описаны линейные матричные уравнения и неравенства вида $HA + A'H < 0$. В конце 40-х и начале 50-х годов XX века в исследованиях А. И. Лурье использовались «разрешающие уравнения Лурье» [11]. Отметим, что сам А. И. Лурье

не использовал аппарат теории матриц, все вычислительные процедуры формулировались в терминах систем скалярных уравнений и неравенств. Необычайно широкое распространение метод линейных матричных неравенств получил в рамках теории абсолютной устойчивости после появления работы В. А. Якубовича [26]. Ему же, по-видимому, принадлежит и термин «метод матричных неравенств». (В известной монографии [35] В. А. Якубович назван отцом теории линейных матричных неравенств, а А. М. Ляпунов — ее дедушкой.) Среди многочисленных работ этого направления отметим [6, 20, 28, 44]. Развитие вычислительной техники потребовало новых методов исследования, ориентированных на численные алгоритмы, легко реализуемые на компьютере. В 80-е годы появились работы Е. С. Пятницкого и его сотрудников, посвященные численной реализации метода линейных матричных неравенств [23, 59], в частности, для их решения был предложен вариант метода эллипсоидов. Существенную роль в дальнейшем развитии этого направления сыграло появление новых эффективных алгоритмов, основанных на методах внутренней точки [54].

Что касается приложений методов выпуклой оптимизации к задачам теории управления, то здесь следует особо отметить монографию [35], в которой впервые были систематически описаны различные задачи теории управления, приводящие к линейным матричным неравенствам. Многочисленные современные приложения линейных матричных неравенств можно найти также в монографии [29]. Они охватывают такие прикладные области как робастное управление, теорию абсолютной устойчивости, H_∞ -оптимизацию, H_2 -оптимизацию, различные вопросы синтеза систем управления. В Интернете свободно распространяется конспект курса лекций, посвященного этой тематике [61]. На русском языке вопросы применения линейных матричных неравенств к задачам робастного управления рассмотрены в [19].

Отметим, что развитие в последнее десятилетие численных методов исследования линейных матричных неравенств не просто добавило новые знания и технические приемы, а фактически изменило язык теории управления. Во многом изменилось содержание понятия «эффективно проверяемое условие». Сложные матричные соотношения, которые еще недавно рассматривались лишь как промежуточные результаты и отправной пункт для дальнейшего анализа, сейчас допускают простое численное исследование. Теперь в терминах этих соотношений можно формулировать теоремы и алгоритмы, без особых усилий сравнивать различные подходы.

Глава 2. Пакет LMILab

Современная версия пакета LMILab была в основном закончена в июне 1995 года [40]. Разработчиками пакета явились П. Гайн (P. Gahinet) и М. Шилали (M. Chilali), сотрудники The MathWorks, Inc., ранее работавшие в INRIA (Франция), а также известные специалисты в области численных методов А. С. Немировский (A. Nemirovski) и А. Лауб (A. Laub). В отличие от других пакетов для решения линейных матричных неравенств, LMILab распространяется на коммерческой основе. В течение последних лет он входит в стандартный дистрибутив системы MATLAB, т.е., приобретая MATLAB, вы, как правило, получаете возможность работать с этим пакетом. При этом LMILab включен в MATLAB не самостоятельно, а внутри более обширного пакета LMI Control Toolbox. Последний распадается на две части: Control Applications и LMILab. Документация к пакету LMI Control Toolbox (в формате Adobe PDF) [40] доступна на сайте компании The Mathworks, Inc. LMILab имеет удобный интерфейс и встроенные решатели, использующие проективный алгоритм Немировского и Нестерова. Разберем работу с функциями пакета подробно.

2.1. Создание системы линейных матричных неравенств

Прежде, чем решать или исследовать линейные матричные неравенства, необходимо их предварительно создать. Каждое неравенство имеет в простейшем случае структуру вида

$$L(x) < R(x). \quad (2.1)$$

Здесь $L(x)$ и $R(x)$ — симметричные матрицы, описывающие левую и правую части неравенства. (Для определенности, пусть их размеры равны $n \times n$.) Математически эта запись означает, что квадратичная форма с матрицей $R(x) - L(x)$ положительно определена, т.е. $y'(R(x) - L(x))y > 0$

для всех ненулевых вещественных векторов y размерности n (апостроф обозначает транспонирование). К сожалению, пакет LMILab не позволяет исследовать нестрогие неравенства вида $L(x) \leq R(x)$ и задачи с ограничениями типа равенств. Для их решения следует использовать другие пакеты, например, LMITOOL или YALMIP.

В более сложном случае неравенство задается в виде

$$M'L(x)M < N'R(x)N, \quad (2.2)$$

где M, N — некоторые заданные постоянные матрицы (внешние множители).

Вектор x представляет собой вектор скалярных переменных в пространстве \mathbb{R}^m , матрицы-функции $L(x)$ и $R(x)$ зависят от него аффинно. Решение неравенства (2.1) относительно x , если оно существует, определяется неоднозначно. (Множество решений (2.1) представляет собой выпуклое множество в \mathbb{R}^m .) Функции LMILab позволяют определить только одно из этих решений.

Создавать матричные неравенства можно двумя различными способами: чисто программно и используя графический интерфейс пользователя (GUI). Разберем оба этих способа, но начнем с создания программного кода.

2.1.1. Функция `setlmis`

Вызов функции `setlmis` начинает описание новой системы матричных неравенств. Если это совершенно новая система, то вызов имеет вид:

```
setlmis([ ])
```

При этом создается «пустая» система линейных матричных неравенств, которую можно дополнять новыми неравенствами. Если мы хотим, чтобы создаваемая система включала в себя ранее созданную систему с именем, например, `lmisys`, то вызов `setlmis` должен выглядеть так:

```
setlmis(lmisys)
```

Здесь `lmisys` содержит внутреннее описание системы неравенств, которое используется для инициализации новой системы.

Дело в том, что большинство функций пакета LMILab работает с «внутренним представлением» системы линейных матричных неравенств в виде одномерного массива специального вида. Именно это внутреннее представление и используется в данном случае. Для получения внутреннего представления по исходному описанию системы служит функция `getlmis` (см. далее).

2.1.2. Функция lmivar

Эта функция служит для создания новой матричной переменной. В записи (2.1) все неизвестные величины включаются в вектор x , однако, это скалярные переменные (decision variables), которые используются вычислительным алгоритмом. В постановке реальной задачи неизвестные величины могут быть скалярами или матрицами различной структуры. Они носят название матричных переменных (matrix variables). Функции пакета автоматически строят по ним вектор скалярных переменных x , т.е. пользователю нет необходимости самому формировать этот вектор. Каждая матричная переменная задается вызовом функции `lmivar`.

В простейшем случае вызов `lmivar` имеет вид:

```
X = lmivar(type,struct)
```

Здесь X — имя создаваемой матричной переменной, входные параметры `type` и `struct` определяют ее структуру. Параметр `type` может принимать значения 1, 2, 3. Значение 1 отвечает симметричной блочно-диагональной матрице, 2 — произвольной прямоугольной матрице, 3 — прямоугольной матрице, описываемой шаблоном.

Параметр `struct` позволяет детализировать содержащуюся в параметре `type` информацию о структуре матрицы X (см. табл. 2.1).

Таблица 2.1.

| type | struct |
|------|---|
| 1 | <code>struct(i, 1)</code> — размер i -го диагонального блока матрицы X ; <code>struct(i, 2)</code> — тип i -го диагонального блока матрицы X : 0 в случае блока вида $t * I$, t — скаляр, I — единичная матрица; 1 в случае произвольного блока; –1 в случае нулевого блока. |
| 2 | <code>struct = [m, n]</code> , если X — матрица размера $m \times n$. |
| 3 | <code>struct</code> — матрица-шаблон того же размера, что и X , ее элементы равны: 0, если $X(i, j) = 0$; $+n$, если элемент $X(i, j)$ равен n -й неизвестной скалярной величине; $-n$, если $-X(i, j)$ равен n -й неизвестной скалярной величине. |

Выходной параметр X равен номеру созданной матричной переменной (т.е. после первого вызова функции `lmivar` он равен 1, после второго — 2 и т.д.).

Для того, чтобы понять смысл переменной типа 3, следует иметь в виду, что одна матричная переменная может содержать несколько скалярных переменных (компонент вектора x из (2.1)). Например, матрица размера 2×2 , имеющая произвольную структуру, содержит четыре неизвестных скалярных параметра. Симметричная матрица такого же размера содержит три неизвестных скалярных параметра, а матрица структуры $t * I$ — один неизвестный параметр. Скалярные неизвестные переменные нумеруются независимо от матричных, также, начиная с единицы.

Более общая форма вызова функции `lmivar` имеет вид:

```
[X,ndec,xdec] = lmivar(type,struct)
```

Выходной параметр `ndec` содержит общее число созданных к данному моменту времени скалярных переменных, `xdec` описывает зависимость X от созданных скалярных переменных (`xdec` совпадает с параметром `struct` в случае `type = 3`).

Примеры.

1. Переменная `eps` определяется как скаляр:

```
eps = lmivar(1,[1 0])
```

2. Переменная `H` представляет собой симметричную $n \times n$ -матрицу:

```
H = lmivar(1,[n 1])
```

3. Матрица `S` симметрична, имеет размеры 4×4 и структуру

$$S = \begin{bmatrix} s_{11} & 0 & 0 & 0 \\ 0 & s_{11} & 0 & 0 \\ 0 & 0 & s_{22} & s_{23} \\ 0 & 0 & s_{23} & s_{33} \end{bmatrix} :$$

```
S = lmivar(1,[2 0; 2 1])
```

4. Матрица `X` — произвольная, размера 5×6 :

```
X = lmivar(2,[5 6])
```

5. Рассмотрим последовательность команд:

```
>>setlmis([ ])
>>[a,n1,x1]=lmivar(1,[2 0])
```

```
a =
1

n1 =
1

x1 =
1     0
0     1
```

Сформированная переменная **a** получает номер 1. Она содержит одну скалярную переменную (ее номер также 1) и имеет структуру диагональной матрицы. Продолжим создание переменных.

```
>> [b,n2,x2]=lmivar(1,[2 1])

b =
2

n2 =
4

x2 =
2     3
3     4
```

Вторая переменная **b** получила номер 2. Она представляет собой симметричную матрицу и содержит три новые скалярные переменные с номерами 2, 3, 4. Общее число созданных скалярных переменных равно четырем. Создадим третью матричную переменную:

```
>> [c,n3,x3]=lmivar(3,[1 0; 0 -4])
```

```
c =
3

n3 =
4

x3 =
1     0
0    -4
```

Матрица с составлена из ранее созданных скалярных переменных (с номерами 1 и 4). Общее число скалярных переменных при этом не увеличилось (осталось равным четырем).

2.1.3. Функция newlmi

После того, как описаны неизвестные переменные, пришла пора формировать линейные матричные неравенства. Описание каждого линейного матричного неравенства обычно начинается вызовом функции `newlmi`. Эта функция сопоставляет неравенству идентификатор (тег), который будет использоваться в дальнейших командах. Схема вызова:

```
lmitag = newlmi
```

Выходной параметр `lmitag` принимает целое значение, равное номеру неравенства в системе.

Команда `newlmi` не обязательна — неравенству можно просто сопоставить порядковый номер в виде целого числа, но использование идентификаторов вместо номеров улучшает читаемость программы.

2.1.4. Функция lmitem

Наряду с `lmivar`, эта функция одна из самых важных для задания линейных матричных неравенств. Она определяет структуру неравенств (т.е. матрицы $R(x)$ и $L(x)$ из (2.1)). Матрицы $L(x)$ и $R(x)$ обычно естественным образом разбиваются на блоки, каждый блок описывается своим вызовом функции `lmitem`. Очень важно учесть следующее: поскольку матрицы $L(x)$ и $R(x)$ симметричны, их блоки, расположенные симметрично относительно главной диагонали, равны. Из каждой такой пары блоков следует описывать только один (нарушение этого правила приводит к ошибке). Например, если

$$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix},$$

где $R_{12} = R_{21}$, то следует описать либо R_{12} , либо R_{21} , но не оба вместе! Не следует также описывать блоки, состоящие из нулей (нули заносятся по умолчанию).

Для описания блока функция `lmitem` вызывается, по крайней мере, один раз (для описания сложных блоков она может вызываться несколько раз). Каждый вызов заносит в блок либо постоянную матрицу-коэффициент, либо слагаемое вида AXB или $AX'B$, где A, B — постоянные

матрицы, X — переменная, созданная с помощью `lmivar`. (Напоминаем, что апостроф обозначает операцию транспонирования.) Кроме того, функция `lmiterm` может служить для задания внешнего множителя, который относится не к отдельному блоку, а ко всей матрице $L(x)$ или $R(x)$.

Функция `lmiterm` не имеет выходных параметров, и ее вызов выглядит следующим образом:

```
lmiterm(termid,A,B,flag)
```

Входные параметры имеют следующий смысл.

Параметр `termid` представляет собой вектор из четырех компонент.

- `termid(1)` определяет, в какое неравенство добавляется член и в какую его часть (правую или левую). Напомним, что рассматриваются только такие неравенства, у которых левая часть строго меньше правой. Значение $+n$ означает работу с левой частью неравенства, имеющего номер n , а значение $-n$ — работу с его правой частью. Вместо числа n обычно используют идентификатор неравенства, полученный с помощью `newlmi`.
- `termid(2 : 3)` определяет координаты блока, в который заносится новый член. Значение `termid(2 : 3) = [i j]` означает блок с координатами (i, j) . Для внешних множителей следует положить `termid(2 : 3) = [0 0]`.
- `termid(4)` определяет тип добавляемого слагаемого:
 $\text{termid}(4) = 0$ — постоянное слагаемое,
 $\text{termid}(4) = X$ — слагаемое вида AXB ,
 $\text{termid}(4) = -X$ — слагаемое вида $AX'B$.

Здесь X — какая-либо переменная, добавленная с помощью `lmivar`.

Параметр `A` задает значение либо внешнего множителя, либо постоянного слагаемого, либо матричного коэффициента в выражениях AXB или $AX'B$.

Параметр `B` определяет правый множитель в выражениях AXB или $AX'B$. В остальных случаях (постоянного слагаемого и внешнего множителя) этот параметр не указывается.

Параметр `flag` — необязательный параметр, который позволяет упростить задание выражения вида $AXB + B'XA'$ в диагональном блоке. При этом достаточно определить слагаемое AXB и придать параметру `flag` значение '`s`'.

Примеры.

1. Зададим неравенство, которое требует положительную определенность симметричной $n \times n$ -матрицы H .

```
H = lmivar(1,[n 1]);
Hpos=newlmi;
lmiterm([-Hpos 1 1 H],1,1);
```

При этом неравенство имеет вид: $H > 0$. С ним связан идентификатор `Hpos`.

2. Пусть A — заданная $n \times n$ матрица, b — n -мерный столбец. Пусть также задана симметричная матрица F порядка $n + 1$. Составим неравенство относительно симметричной $n \times n$ матрицы H вида

$$\begin{bmatrix} HA + A'H & Hb \\ b'H & 0 \end{bmatrix} < F.$$

Составим фрагмент программы:

```
H = lmivar(1,[n 1]);
lmi1=newlmi;
lmiterm([lmi1 1 1 H],1,A,'s');
lmiterm([lmi1 1 2 H],1,b);
lmiterm([-lmi1 1 1 0],F);
```

Неравенству сопоставлен идентификатор `lmi1`.

3. Пусть матрицы коэффициентов определены так же, как в предыдущем примере. Запишем неравенство

$$\begin{bmatrix} A'HA - H & A'Hb \\ b'HA & b'Hb \end{bmatrix} < \begin{bmatrix} G & g \\ g' & \Gamma \end{bmatrix},$$

где G, g, Γ — подматрицы подходящих размеров, матрицы G и Γ — симметричные. Имеем:

```
H = lmivar(1,[n 1]);
lmi2=newlmi;
lmiterm([lmi2 1 1 H],A',A);
lmiterm([lmi2 1 1 H],1,-1);
lmiterm([lmi2 1 2 H],A',b);
lmiterm([lmi2 2 2 H],b',b);
lmiterm([-lmi2 1 1 0],G);
lmiterm([-lmi2 1 2 0],g);
lmiterm([-lmi2 2 2 0],Gamma);
```

Заметим, что для описания блока, равного $A'HA - H$, функция `lmiterm` была вызвана два раза.

2.1.5. Функция `getlmis`

Функция `getlmis` вызывается на заключительном этапе формирования системы линейных матричных неравенств. Она служит для получения внутреннего представления системы в виде одномерного массива специального вида. Вызов функции `getlmis` имеет вид:

```
lmisys = getlmis
```

Выходной параметр `lmisys` может быть передан информационным функциям или решающим функциям (`solvers`).

Таким образом, общая схема формирования системы линейных матричных неравенств состоит из вызовов следующих функций:

- `setlmis`,
- `lmivar`,
- `newlmi`,
- `lmiterm`,
- `getlmis`.

Функции `setlmis` и `getlmis` вызываются один раз, в начале и в конце формирования системы неравенств. Система может включать несколько различных неравенств, поэтому функция `newlmi` может вызываться несколько раз.

2.1.6. Интерактивное создание системы линейных матричных неравенств. Команда `lmiedit`

Эта команда служит для интерактивного задания линейных матричных неравенств. При ее вызове появляется диалоговое окно. Подсказки по заполнению его отдельных полей можно получить, нажимая кнопки «`help`». Использование `lmiedit` значительно облегчает процесс создания программного кода. Тем не менее, мы рекомендуем пользователю прежде, чем воспользоваться этой командой, потренироваться в непосредственном написании кодов вызова функций `lmivar` и `lmiterm`.

Для создания новой системы неравенств выполните следующую последовательность действий.

- В строке ввода с меткой «`name of LMI system:`» задайте идентификатор системы неравенств.

- Верхние поля редактирования служат для определения матричных переменных системы. Выберите переключатель «describe the matrix variables». В поле редактирования с меткой «variable name» занесите имя матричной переменной. В поле с заголовком «type (S/R/G)» занесите одну из трех букв — S, R или G. (Сокращения от английских слов *symmetric* — симметричный, *rectangular* — прямоугольный, *general* — общий.) Эти буквы соответствуют значениям 1–3 входного параметра *type* функции *lmivar* (см. выше). В поле «structure» заносится значение параметра *struct* функции *lmivar*. Например, значения

X S [1,0;1,0;1,0]

определяют диагональную матрицу *X* порядка 3. Значения

Y R [2,3]

определяют прямоугольную матрицу *Y* размеров 3×2 . Теперь, если выбрать переключатель «view commands», вы увидите фрагмент кода с вызовами функций *setlmis* и *lmivar*, отвечающими заполненным вами полям:

```
setlmis([ ]);
X=lmivar(1,[1,0;1,0;1,0]);
Y=lmivar(2,[2,3]);
```

- Нижняя часть окна служит для определения самих линейных матричных неравенств. При этом неравенства можно задавать в более простой и понятной форме, чем при непосредственном вызове *lmiterm*. Выберите переключатель «describe the LMIs as MATLAB expressions» (т.е. «описать линейные матричные неравенства как выражения MATLAB»). В поле редактирования выпишите матричные неравенства, используя обычный синтаксис MATLAB. Например:

```
[A'*X+X*A+C'*Y*C      X*B;      B'*X      -Y]   <   0
X > 0
Y < 1
```

В выражениях, задающих матричные неравенства, имена матричных переменных нельзя помещать внутри круглых скобок. Выбрав

переключатель «view commands», вы можете посмотреть получившийся программный код. Так, для приведенного выше примера получим:

```
lmiterm([1 1 1 X],A',1,'s');           % LMI #1: A'*X+X*A
lmiterm([1 1 1 Y],C',C);               % LMI #1: C'*Y*C
lmiterm([1 2 1 X],B',1);               % LMI #1: B'*X
lmiterm([1 2 2 Y],1,-1);              % LMI #1: -Y

lmiterm([-2 1 1 X],1,1);             % LMI #2: X

lmiterm([3 1 1 Y],1,1);             % LMI #3: Y
lmiterm([-3 1 1 0],1);              % LMI #3: 1

lmi1=getlmis;
```

В нижней части окна расположен ряд командных кнопок.

- Кнопка **save** сохраняет созданное описание системы неравенств в виде строки MATLAB. Пользователю предлагается ввести имя строки во вспомогательном окне.
- Кнопка **load** загружает описание системы неравенств, ранее сохраненное в строке (требуется указать имя строки).
- Кнопка **write** сохраняет сгенерированный командой программный код в текстовом файле (требуется указать имя файла).
- Кнопка **read** вызывает чтение программного кода, сохраненного в файле.
- Кнопка **create** служит для формирования внутреннего (векторного) описания системы линейных матричных неравенств. Это описание сохраняется в массиве, имя которого совпадает с именем системы неравенств. Полученный массив может быть передан другим функциям пакета.
- Кнопка **clear all** вызывает очистку всех полей редактирования.
- Кнопка **close** закрывает окно.

2.2. Получение информации о системе

Пакет LMILab содержит ряд вспомогательных функций, задача которых — извлечение полезных сведений из внутреннего представления системы матричных неравенств. Как и в случае создания системы, эти функции существуют в интерактивном и неинтерактивном вариантах.

2.2.1. Функция `lminbr`

Исходя из внутреннего представления системы, функция `lminbr` возвращает число входящих в нее неравенств. Вызов функции:

```
nlmis = lminbr(lmisys)
```

2.2.2. Функция `matnbr`

В простейшем случае

```
nmvars = matnbr(lmisys)
```

Эта функция возвращает число матричных переменных, используемых в рассматриваемой системе. В случае вызова с двумя выходными параметрами

```
[nmvars,varid] = matnbr(lmisys)
```

дополнительно возвращается список идентификаторов матричных переменных (в выходном параметре `varid`).

2.2.3. Функция `decnbr`

Возвращает число неизвестных скалярных переменных (decision variables) в системе. Вызов функции:

```
ndecv = decnbr(lmisys)
```

2.2.4. Команда `lmiinfo`

Команда `lmiinfo` позволяет в интерактивном режиме получить полную информацию о системе неравенств, исходя из ее внутреннего описания. В отличие от `lmiedit`, `lmiinfo` использует не графический, а текстовый интерфейс. Вызов команды имеет вид:

```
lmiinfo(lmisys)
```

Здесь `lmisys` — внутреннее представление системы неравенств.

Пусть, например, A — заданная матрица размера 2×2 , b — заданный двумерный вектор-столбец. Зададим переменную симметричную 2×2 матрицу H и систему неравенств

$$HA + A'H < 0, \quad H > 0.$$

Сохраним внутреннее описание системы в переменной `l1`. После вызова

```
lmiinfo(l1)
```

на экране появится текст:

```
>> lmiinfo(l1)
```

```
LMI      ORACLE  
-----  
This is a system of 2 LMI(s) with 1 matrix variables  
  
Do you want information on  
    (v) matrix variables      (l) LMIs      (q) quit  
?>
```

Нажатие клавиши «v» служит для просмотра информации о матричных переменных системы, клавиши «l» — о структуре неравенств и, наконец, нажатие «q» вызывает окончание диалога.

Нажмем клавишу «v». Получим запрос:

```
Which variable matrix (enter its index k between 1 and 1) ?
```

Здесь нас просят ввести порядковый номер матричной переменной. Поскольку в данном случае переменная всего одна, то выбирать не из чего — вводим единицу. Диалог продолжается:

```
X1 is a 2x2 symmetric block diagonal matrix  
its (1,1)-block is a full block of size 2  
-----
```

```
This is a system of 2 LMI with 1 variable matrices  
  
Do you want information on  
    (v) matrix variables      (l) LMIs      (q) quit  
?>
```

Описание переменной получено. Теперь рассмотрим сами линейные матричные неравенства. Для этого введем «l». Получим запрос:

```
Which LMI (enter its number k between 1 and 2) ?
```

Здесь нам нужно указать номер неравенства (1 или 2). Введем 1. Получаем:

```
This LMI is of the form
L(x) < 0
where the inner factor(s) has 1 diagonal block(s)
```

```
Do you want info on the left inner factor ?
```

```
(w) whole factor      (b) only one block
(o) other LMI         (t) back to top level
```

```
?>
```

Нам предлагают получить информацию о левой части неравенства (матрице $L(x)$). В ответ мы можем ввести: «w» — информация обо всей матрице, «b» — информация об отдельном блоке матрицы, «o» — информация о другом неравенстве, «t» — вернуться на предыдущий уровень диалога. Ответим «w». Получим:

```
Left inner factor:
```

```
block (1,1): A1*X1 + X1*A1'
```

```
(w) whole factor      (b) only one block
(o) other LMI         (t) back to top level
```

```
?>
```

Если ввести «b», то нас попросят указать координаты блока:

```
Enter the block coordinates (1 <= i,j <= 1) :
i = 1
j = 1
```

```
This block is of the form
```

```
A1*X1 + X1*A1'
```

```
?> (w) whole factor      (b) only one block  
     (o) other LMI          (t) back to top level
```

Дальнейшие действия достаточно очевидны — вы можете изучить структуру вашей системы неравенств и убедиться, что она не содержит ошибок.

2.2.5. Команда decinfo

Эта команда позволяет исследовать зависимость матричных переменных (matrix variables) и скалярных переменных (decision variables). Она может выполняться в интерактивном и неинтерактивном режиме. Форма вызова для работы в интерактивном режиме:

```
decinfo(lmisys)
```

где `lmisys` — внутреннее представление системы неравенств. Для примера, рассмотренного в предыдущем разделе, получим:

```
>> decinfo(l1)
```

```
There are 3 decision variables labeled x1 to x3 in this  
problem.
```

```
Matrix variable Xk of interest (enter k between 1 and 1,  
or 0 to quit):
```

```
?>
```

Далее следует указать номер матричной переменной. Введем 1. Получаем:

```
The decision variables in X1 are among {x1,...,x3}.  
Their entry-wise distribution in X1 is as follows  
(0,j,-j stand for 0,xj,-xj, respectively):
```

```
X1 :
```

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |

```
*****
```

```
Matrix variable Xk of interest (enter k between 1 and 1,  
or 0 to quit):  
?>
```

Получены данные о структуре матричной переменной X_1 . В данном случае видно, что это симметричная матрица, составленная из скалярных переменных с номерами 1, 2, 3. В общем случае матрица описания структуры может содержать значения: 0 (нулевой элемент), n или $-n$ (здесь n — целое число — номер скалярной переменной).

Неинтерактивная форма вызова функции следующая:

```
decx = decinfo(lmisys,x)
```

Здесь x — имя матричной переменной, $decx$ — матрица описания ее структуры.

Для рассмотренного выше примера:

```
>> decH=decinfo(l1,H)
```

```
decH =  
1 2  
2 3
```

2.3. Взаимные преобразования матричных и скалярных переменных

2.3.1. Функция dec2mat

Функции-решатели (solvers) возвращают результат своей работы в виде массива скалярных переменных. Функция `dec2mat` позволяет извлечь из него значения матричных переменных. Вызов функции имеет вид:

```
X = dec2mat(lmisys,decvars,k)
```

Здесь `lmisys` содержит внутреннее представление системы неравенств, `decvars` — массив (вектор) скалярных переменных, k — номер извлекаемой матричной переменной (в качестве k можно использовать идентификатор этой переменной). Функция возвращает искомую переменную в виде матрицы. Примеры использования функции `dec2mat` будут даны в дальнейшем.

2.3.2. Функция mat2dec

Функция `mat2dec` выполняет обратную операцию — формирует из последовательности матричных переменных одномерный массив скалярных переменных. Вызов функции:

```
decvars = mat2dec(lmisys,X1,X2,X3,...)
```

Здесь X_1, X_2, X_3, \dots — последовательность матричных переменных (не более двадцати). К моменту вызова функции их значения должны быть определены (иначе возникает ошибка). Функция возвращает вектор скалярных переменных.

2.3.3. Функция defcx

Функция `defcx` носит чисто вспомогательный характер — она используется для подготовки данных перед применением функции `mincx` (см. далее). Вызов функции выглядит следующим образом:

```
[V1,...,Vk] = defcx(lmisys,n,X1,...,Xk)
```

Здесь `lmisys` — внутреннее представление системы линейных матричных неравенств, X_1, \dots, X_k — последовательность матричных переменных, n — номер скалярной переменной. На выходе функции — последовательность матриц тех же размеров, что и X_1, \dots, X_k . Эти матрицы составляются следующим образом. Матрица V_k получается из матрицы X_k , в которой n -я скалярная переменная равна единице, а все остальные переменные равны нулю. Таким образом, можно явно проследить все вхождения скалярной переменной в матричные переменные.

Пример. Определим систему `lmisys` с помощью последовательности команд:

```
>> setlmis([ ]);  
>> P=lmivar(1,[2 1]);  
>> Q=lmivar(3,[1 0; 0 -3]);  
>> lmisys=getlmis;
```

Матричные переменные P и Q имеют структуру

$$P = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \end{bmatrix}, \quad Q = \begin{bmatrix} x_1 & 0 \\ 0 & -x_3 \end{bmatrix},$$

где x_1, x_2, x_3 — скалярные переменные. Теперь три раза вызовем функцию `defcx`:

```
>> [U V]=defcx(lmisys,1,P,Q)
```

```
U =
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

```
V =
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

```
>> [U V]=defcx(lmisys,2,P,Q)
```

```
U =
```

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
V =
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
>> [U V]=defcx(lmisys,3,P,Q)
```

```
U =
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

```
V =
```

$$\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}$$

```
>>
```

Полученный результат очевидно согласуется с представлением

$$P = x_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad Q = x_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + x_3 \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}.$$

Для того, чтобы использовать функцию `mincx`, требуется предварительно преобразовать целевую функцию. Ее необходимо представить в виде линейной формы $c'x$, где x — вектор скалярных переменных. Функция `defcx` позволяет достаточно просто найти коэффициенты этой формы, т.е. найти вектор c . Пусть матричная переменная H представлена в виде

$$H = \sum_{j=1}^m x_j H_j,$$

где x_j — скалярные переменные, H_j — постоянные квадратные матрицы. Тогда выражение вида $b' H b$, где b — заданный вектор, может быть записано как

$$b' H b = \sum_{j=1}^m x_j b' H_j b,$$

т.е. $b' H b = c' x$, где $c_j = b' H_j b$.

Аналогично, в силу линейности следа матрицы, выражение $\text{tr}(BH)$, где B — постоянная матрица, можно представить в виде $\text{tr}(BH) = c' x$, где $c_j = \text{tr}(BH_j)$.

Пример. Представим в виде линейной формы вида $c' x$ выражение $\text{tr } P + b' Q b$, где P и Q — матричные переменные, b — заданный вектор. Коэффициенты вектора c могут быть найдены следующим образом:

```
n = decnbr(lmisys);
c = zeros(n,1);
for j = 1:n
    [Pt,Qt] = defcx(lmisys,j,P,Q);
    c(j) = trace(Pt) + b'*Qt*b;
end
```

Вторая строка этого кода выполняет предварительное выделение памяти для массива (preallocation), что ускоряет работу цикла.

Пример. Пусть требуется минимизировать значение матричной переменной `eps`, которая описывает некоторое скалярное числовое значение. Сформируем коэффициенты вектора c :

```
n = decnbr(lmisys);
c = zeros(n,1);
for j = 1:n
    c(j) = defcx(lmisys, j, eps);
end
```

2.4. Исследование системы линейных матричных неравенств

Функции этой группы — самые главные в пакете, собственно ради них пакет и разрабатывался. В качестве входного параметра они используют

внутреннее представление системы линейных неравенств (в виде массива). Примеры применения этих функций довольно сложны и содержатся в конце главы.

2.4.1. Функция feasp

Функция **feasp** решает задачу нахождения какого-либо решения x системы неравенств вида (2.1) (feasibility problem). Попутно функция проверяет, имеет ли система неравенств вообще какие-либо решения. По исходному неравенству (2.1) составляется вспомогательная задача выпуклого программирования:

$$t \rightarrow \min \quad \text{при} \quad L(x) < R(x) + tI,$$

где t — скалярный параметр. (Конечно, математически более корректно здесь говорить об инфимуме, а не о минимуме.) Для того, чтобы неравенство (2.1) было разрешимо относительно x , необходимо и достаточно, чтобы решение t оптимизационной задачи было строго отрицательным. С помощью итерационного алгоритма функция **feasp** формирует строго убывающую последовательность приближений t_1, t_2, \dots

Вызов функции **feasp** имеет вид:

```
[tmin,xfeas] = feasp(lmisys,options,target)
```

Ее выходные значения: **tmin** — минимальное найденное значение t , **xfeas** — оптимальное значение вектора x . Таким образом, результат получается в виде вектора скалярных переменных. Чтобы получить из него матричные переменные, следует применить функцию **dec2mat**. По умолчанию на экран выводится вся последовательность t_1, t_2, \dots . Рассмотрим входные параметры функции.

Параметр **lmisys** содержит описание системы линейных матричных неравенств (внутреннее представление) в виде одномерного массива.

Параметр **options** представляет собой одномерный массив, который может включать до пяти элементов. Если **options(i) = 0**, используется значение по умолчанию. Правила задания элементов массива приведены в табл. 2.2.

Параметр **target** определяет условие окончания итерационного процесса. Вычисления прекращаются, когда $t < \text{target}$. По умолчанию **target = 0**.

Таблица 2.2.

| | |
|-------------------------|--|
| <code>options(1)</code> | Не используется. |
| <code>options(2)</code> | Максимальное число итераций (по умолчанию = 100). |
| <code>options(3)</code> | Вещественное значение R . Предполагается, что $\ x\ < R$, где $\ \cdot\ $ — евклидова норма вектора. По умолчанию $R = 10^9$. Значение $R < 0$ означает отсутствие каких-либо ограничений. |
| <code>options(4)</code> | Целое число $L > 1$. Вычисления прекращаются, когда за L последних итераций величина t уменьшилась не более, чем на 1%. По умолчанию $L = 10$. |
| <code>options(5)</code> | Если придать этому параметру ненулевое значение, трассировка результатов промежуточных итераций отключается. |

2.4.2. Функция `mincx`

Функция `mincx` решает оптимизационную задачу

$$c'x \rightarrow \min \quad \text{при} \quad L(x) < R(x),$$

где x — вектор скалярных переменных, c — заданный вектор-столбец.

Вызов функции выглядит следующим образом:

```
[copt,xopt] = mincx(lmisys,c,options,xinit,target)
```

Здесь `lmisys` — внутреннее описание системы неравенств, задающей ограничения $L(x) < R(x)$, c — вектор-столбец того же размера, что и x . Для получения c обычно используется функция `defcx`.

Как и для функции `feasp`, параметр `options` не обязателен и представляет собой одномерный массив, который может включать до пяти элементов (см. табл. 2.3).

Параметр `xinit` задает начальное приближение оптимальной точки x . Если начальное приближение не задано, его можно положить пустым (т.е. указать пустые квадратные скобки). Если этот параметр не удовлетворяет системе ограничений, то он игнорируется.

Параметр `target` — необязательный параметр, который определяет правило остановки вычислений. Итерации прекращаются, если найдено значение x , которое удовлетворяет ограничениям $L(x) < R(x)$ и такое, что $c'x < \text{target}$. По умолчанию `target = -1020`.

Функция возвращает два значения: `copt` — глобальный минимум целевой функции $c'x$ и `xopt` — оптимальное значение вектора скалярных

Таблица 2.3.

| | |
|-------------------------|--|
| <code>options(1)</code> | Относительная точность вычисления оптимального значения $c'x$ (по умолчанию — 0.01). |
| <code>options(2)</code> | Максимальное число итераций (по умолчанию — 100). |
| <code>options(3)</code> | Вещественное значение R . Предполагается, что присутствует ограничение $\ x\ < R$, где $\ \cdot\ $ — евклидова норма вектора. По умолчанию $R = 10^9$. Значение $R < 0$ означает отсутствие каких-либо ограничений. |
| <code>options(4)</code> | Целое число $L > 1$. Вычисления прекращаются, когда за L последних итераций величина $c'x$ уменьшилась не более, чем на относительную величину, указанную в параметре <code>options(1)</code> . По умолчанию $L = 10$. |
| <code>options(5)</code> | Если придать этому параметру ненулевое значение, трассировка результатов промежуточных итераций отключается. |

переменных x . Для того, чтобы извлечь из него матричные переменные, требуется применить функцию `dec2mat`. В случае, если система ограничений не имеет решения (множество допустимых переменных пусто), оба выходных параметра принимают пустое значение [].

2.4.3. Функция `gevp`

Функция `gevp` предназначена для оценки обобщенных собственных чисел (generalized eigenvalue minimization problem). Пусть заданы симметричные матрицы $C(x)$, $A_j(x)$, $B_j(x)$, $j = 1, \dots, m$, зависящие от вектора скалярных параметров x . Функция решает оптимизационную задачу:

$$\begin{aligned} t &\rightarrow \min, \\ C(x) &< 0, \quad 0 < B_j(x), \quad A_j(x) < tB_j(x), \quad j = 1, \dots, m. \end{aligned}$$

Заметим, что в частном случае, когда $B_j(x) = I$, неравенство $A_j(x) < tI$ дает оценку максимального собственного значения симметричной матрицы $A_j(x)$.

Вызов функции выглядит следующим образом:

```
[tmin,xopt] = gevp(lmisys,m,options,t0,x0,target)
```

Таблица 2.4.

| | |
|-------------------------|---|
| <code>options(1)</code> | Относительная точность вычисления оптимального значения t (по умолчанию — 0.01). |
| <code>options(2)</code> | Максимальное число итераций (по умолчанию — 100). |
| <code>options(3)</code> | Вещественное значение R . Предполагается, что $\ x\ < R$, где $\ \cdot\ $ — евклидова норма вектора. По умолчанию $R = 10^8$. Значение $R < 0$ означает отсутствие каких-либо ограничений. |
| <code>options(4)</code> | Целое число $L > 1$. Вычисления прекращаются, когда за L последних итераций величина t уменьшилась не более, чем на относительную величину, указанную в параметре <code>options(1)</code> . По умолчанию $L = 5$. |
| <code>options(5)</code> | Если придать этому параметру ненулевое значение, трассировка результатов промежуточных итераций отключается. |

Здесь `lmisys` содержит внутреннее описание системы неравенств, задающей ограничения. При определении этой системы неравенства, включающие параметр t , должны определяться последними. Параметр `m` задает число линейных неравенств, включающих t . Сам параметр t как-либо описывать и явно включать в неравенства не нужно. Программа сама отсчитывает m последних неравенств и добавит в них t , поэтому последовательность описания неравенств очень важна. Заметим, что неравенства, задающие ограничения $B_j(x) > 0$, обязательно должны присутствовать в системе. Если их нет, функция может вернуть результат, но он достаточно произведен и, как правило, представляет собой довольно большое число.

Как и раньше, `options` — необязательный массив из пяти элементов (см. табл. 2.4).

Остальные входные параметры необязательны: `t0` и `x0` задают начальные приближения для t и x (игнорируются, если не удовлетворяют системе ограничений), `target` задает правило остановки — вычисления прекращаются, если $t < \text{target}$ (по умолчанию `target = -10^5`).

Функция возвращает значения: `tmin` — минимум величины t и `xopt` — оптимальное значение вектора скалярных переменных x . Для того, чтобы извлечь из него матричные переменные, требуется применить функцию `dec2mat`.

2.4.4. Функция basiclmi

Эта функция решает частную задачу исследования линейного матричного неравенства

$$M + P'XQ + Q'X'P < 0,$$

где M, P, Q — заданные матрицы, причем матрица M — симметричная, матрицы P и Q — прямоугольные, X — неизвестная прямоугольная матрица. Можно показать, что это неравенство разрешимо тогда и только тогда, когда

$$P'_0MP_0 < 0, \quad Q'_0MQ_0 < 0,$$

где столбцы матриц P_0 и Q_0 образуют базисы в нуль-пространствах матриц P и Q соответственно.

Вызов функции имеет вид:

```
X = basiclmi(M,P,Q,options)
```

Если неравенство неразрешимо, то выходное значение $X = []$. Параметр `options` не обязательный: если положить его равным '`Xmin`', то ищется значение X с наименьшей нормой. Здесь имеется в виду норма Фробениуса, т.е.

$$\|X\| = \left(\sum_i (X'X)_{ii} \right)^{1/2}.$$

2.5. Оценивание линейных матричных неравенств

2.5.1. Функция evalmli

Эта функция делает подстановку заданного значения вектора скалярных параметров x в систему линейных матричных неравенств и возвращает результат в виде внутреннего описания (одномерного массива). Вызов функции имеет вид:

```
evalsys = evalmli(lmisys,decvars)
```

Здесь `lmisys` описывает внутреннее представление системы матричных неравенств, `decvars` — вектор скалярных переменных x (обычно он является результатом работы функции-решателя). Для того, чтобы извлечь из описания `evalsys` полезные данные, следует воспользоваться функцией `showlmi`.

2.5.2. Функция `showlmi`

Эта функция вычисляет левую и правую части матричного неравенства после подстановки определенных значений скалярных переменных. Ее вызов:

```
[lhs,rhs]=showlmi(evalsys,n)
```

Здесь `evalsys` — внутреннее представление системы неравенств, полученное в результате вызова функций `evallmi` или `setmvar`, `n` — порядковый номер неравенства в системе. Результатом работы являются матрицы $L(x)$ и $R(x)$, вычисленные в точке x , в которой выполняется оценка (эта точка опосредованно содержится во входном параметре `evalsys`).

2.6. Модификация системы линейных матричных неравенств

2.6.1. Функция `dellmi`

Функция удаляет из системы неравенство с заданным номером. Вызов функции имеет вид:

```
newsys = dellmi(lmisys,lmid)
```

Здесь `lmid` — порядковый номер или идентификатор линейного матричного неравенства, входящего в систему (проще всего воспользоваться значением, полученным при вызове `newlmi`). Матричные переменные, которые входят только в удаляемое неравенство, также автоматически удаляются. Результат работы функции — измененное внутреннее представление системы.

2.6.2. Функция `delmvar`

Функция удаляет из системы матричную переменную. Вызов функции:

```
newsys = delmvar(lmisys,k)
```

Здесь `k` — порядковый номер или идентификатор удаляемой матричной переменной (используйте значение, которое возвращает функция `lmivar` при создании переменной). Результат работы — описание обновленной системы.

2.6.3. Функция setmvar

Функция подставляет в систему неравенств заданное значение указанной переменной. Вызов функции имеет вид:

```
newsys = setmvar(lmisys,k,X)
```

Здесь k — номер или идентификатор матричной переменной, X — ее значение. Во все части неравенств, содержащие k -ю матричную переменную подставляется значение X . Для определения величины k используйте значение, которое возвращает функция lmivar при создании переменной. Функция setmvar не меняет идентификаторы оставшихся переменных, т.е. к оставшимся переменным можно по-прежнему обращаться, используя их первоначальные идентификаторы.

2.7. Примеры работы с пакетом

2.7.1. Круговой критерий абсолютной устойчивости

Применим пакет для исследования одной из классических задач теории абсолютной устойчивости.

Рассмотрим нелинейную систему управления

$$\frac{dx}{dt} = Ax + b\xi, \quad \sigma = c'x, \quad (2.3)$$

$$\xi = \varphi(\sigma, t). \quad (2.4)$$

Здесь A — постоянная $m \times m$ -матрица, b и c — постоянные m -мерные векторы-столбцы. В терминах преобразований Лапласа $\tilde{\sigma}$, $\tilde{\xi}$ связаны соотношением

$$\tilde{\sigma} = -W(s)\tilde{\xi},$$

где $W(s) = c'(A - sI_m)^{-1}b$ — передаточная функция линейной части системы от входа $-\xi$ к выходу σ . Нелинейная часть системы (2.4) описывается нелинейной функцией (в более общем случае — нелинейным оператором) $\varphi(\sigma)$. Пусть она принадлежит классу нелинейностей

$$\mu_1 \leq \frac{\varphi(\sigma, t)}{\sigma} \leq \mu_2, \quad \forall \sigma, \forall t, \quad (2.5)$$

где μ_1, μ_2 — заданные вещественные числа, $\mu_1 < \mu_2$. Очевидно при этом $\varphi(0, t) \equiv 0$ и система (2.3), (2.4) имеет нулевое состояние равновесия $x(t) \equiv 0$. Система (2.3), (2.4) называется абсолютно устойчивой в классе нелинейностей (2.5), если ее нулевое состояние равновесия равномерно

экспоненциально устойчиво в целом: существуют постоянные $M > 0$, $\varepsilon > 0$ такие, что $\|x(t)\| \leq M\|x(t_0)\| \exp(-\varepsilon(t-t_0))$ при всех $t \geq t_0$, причем M и ε не зависят от функции φ .

В терминах линейных матричных неравенств круговой критерий может быть сформулирован следующим образом.

Теорема 2.1. *Пусть существуют симметричная положительно определенная матрица H и число $\delta > 0$, удовлетворяющие неравенству*

$$2x'H(Ax+b\xi) + (\mu_2 c'x - \xi)(\xi - \mu_1 c'x) < -\delta(\|x\|^2 + |\xi|^2), \quad \forall x \in \mathbb{R}^m, \forall \xi \in \mathbb{R}. \quad (2.6)$$

Тогда система (2.3), (2.4) абсолютно устойчива в классе нелинейностей (2.5).

Таким образом, проверка кругового критерия сводится к проверке разрешимости системы линейных матричных неравенств относительно симметричной матрицы H :

$$\begin{bmatrix} HA + A'H & Hb \\ b'H & 0 \end{bmatrix} < \begin{bmatrix} \mu_1 \mu_2 cc' & -\frac{1}{2}(\mu_1 + \mu_2)c \\ -\frac{1}{2}(\mu_1 + \mu_2)c' & 1 \end{bmatrix}, \quad H > 0. \quad (2.7)$$

Приведенная ниже функция `circle` решает эту задачу. Она использует функцию пакета LMILab `feasp`. Текст функции может быть помещен в файл `circle.m`.

```
function [flag,H0] = circle(A,b,c,mu1,mu2)
%Circle criterion
sz=size(A);
flag=0;
H0=[ ];
setlmis([ ]);
H=lmivar(1,[sz(1) 1]);
MLMI=newlmi;
lmiterm([MLMI 1 1 H],1,A,'s');
lmiterm([MLMI 1 2 H],1,b);
lmiterm([-MLMI 1 1 0],mu1*mu2*c*c');
lmiterm([-MLMI 1 2 0],-0.5*(mu1+mu2)*c);
lmiterm([-MLMI 2 2 0],1);
Hpos=newlmi;
lmiterm([-Hpos 1 1 H],1,1);
lmisys=getlmis;
[tmin,x0]=feasp(lmisys,[0 100 1e9 10 1]);
```

```

if tmin<0
    flag=1;
    H0=dec2mat(lmisy, x0, 1);
end

```

Выходные параметры функции: `flag` равен 1, если условия критерия выполнены, и 0, если не выполнены, `H0` — значение матрицы H (если задача не имеет решения, этот параметр получает пустое значение).

Приведем пример вызова функции `circle`:

```

clc;
mu1=-0.1;
mu2=0.1;
numerator=[1];
denominator=[1 2 0.5];
w=tf(numerator,denominator);
sys=ss(w);
[A,b,c,d]=ssdata(sys);
c=-c';
[flag,H]=circle(A,b,c,mu1,mu2);
if flag==1
    disp('Problem is feasible');
    H
else
    disp('Problem is not feasible');
end

```

Здесь для наглядности входные данные заложены непосредственно в код (для большей общности их следовало бы ввести). Задаются параметры μ_1 и μ_2 , а также числитель (`numerator`) и знаменатель (`denominator`) передаточной функции $W(s)$ (перечисляются коэффициенты в порядке убывания степеней). Они объединяются в передаточную функцию с помощью функции `tf` пакета Control System Toolbox. В данном примере получаем

$$W(s) = \frac{1}{s^2 + 2s + 0.5}.$$

С помощью функции `ssdata` того же пакета по передаточной функции строится ее минимальная реализация A, b, c, d . Обратим внимание на некоторые «подводные камни». Функция `ssdata` реализует передаточную функцию в виде

$$W(s) = c(sI - A)^{-1}b + d.$$

В данном примере получаем:

$$A = \begin{bmatrix} -2.0000 & -0.1250 \\ 4.0000 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0.5000 \\ 0 \end{bmatrix}, \quad c = [0 \quad 0.5000], \quad d = 0.$$

Таким образом, c получается как вектор-строка, тогда как в теории абсолютной устойчивости здесь чаще используется вектор-столбец (см., например, [20]). Во вторых, используемая в теории абсолютной устойчивости передаточная функция отличается от традиционной передаточной функции знаком (это функция от $-\xi$ к σ , а не от ξ к σ). Эти два обстоятельства объясняют появление оператора $c=-c'$.

Результат работы тестового примера следующий:

Problem is feasible

```
H =
0.7240    0.2357
0.2357    0.1671
```

>>

2.7.2. Критерий Попова абсолютной устойчивости

Вновь рассмотрим систему (2.3), (2.4). На этот раз предполагаем, что $\xi = \varphi(\sigma)$, т.е. функция φ не зависит от времени. Рассмотрим класс нелинейностей

$$0 \leq \frac{\varphi(\sigma)}{\sigma} \leq \mu, \quad \forall \sigma, \quad (2.8)$$

где μ — некоторое заданное положительное число. В терминах линейных матричных неравенств критерий В. М. Попова выглядит следующим образом.

Теорема 2.2. *Пусть существуют симметричная положительно определенная матрица H , а также числа θ (произвольного знака) и $\delta > 0$, удовлетворяющие неравенству*

$$2x'H(Ax + b\xi) + \theta\xi c'(Ax + b\xi) + (\mu c'x - \xi)\xi < -\delta(\|x\|^2 + |\xi|^2), \quad \forall x \in \mathbb{R}^m, \quad \forall \xi \in \mathbb{R}.$$

Тогда система (2.3), (2.4) абсолютно устойчива в классе нелинейностей (2.8).

Таким образом, имеем две переменные величины H и θ , а также систему неравенств

$$\begin{bmatrix} HA + A'H & Hb \\ b'H & 0 \end{bmatrix} + \begin{bmatrix} 0 & \frac{1}{2}\theta A'c \\ \frac{1}{2}\theta c'A & \theta c'b \end{bmatrix} < \begin{bmatrix} 0 & -\frac{1}{2}\mu c \\ -\frac{1}{2}\mu c' & 1 \end{bmatrix}, \quad H > 0. \quad (2.9)$$

Приведем текст функции для проверки критерия Попова:

```
function [flag,H0,theta0] = cpopov(A,b,c,mu)
%Popov criterion
sz=size(A);
flag=0;
H0=[ ];
theta0=[ ];
setlmis([ ]);
H=lmivar(1,[sz(1) 1]);
theta=lmivar(1,[1 1]);
MLMI=newlmi;
lmiterm([MLMI 1 1 H],1,A,'s');
lmiterm([MLMI 1 2 H],1,b);
lmiterm([MLMI 1 2 theta],A',0.5*c);
lmiterm([MLMI 2 2 theta],1,c'*b);
lmiterm([-MLMI 1 2 0],-0.5*mu*c);
lmiterm([-MLMI 2 2 0],1);
Hpos=newlmi;
lmiterm([-Hpos 1 1 H],1,1);
lmisys=getlmis;
[tmin,x0]=feasp(lmisys,[0 100 1e9 10 1]);
if tmin<0
    flag=1;
    H0=dec2mat(lmisys,x0,H);
    theta0=dec2mat(lmisys,x0,theta);
end
```

Выходной параметр `flag` имеет тот же смысл, что и в предыдущем примере, `H0` и `theta0` — решения системы неравенств.

Вызов этой функции можно выполнить следующим образом:

```
clc;
mu=0.1;
numerator=[1];
```

```
denominator=[1 2 0.5];
w=tf(numerator,denominator);
sys=ss(w);
[A,b,c,d]=ssdata(sys);
c=-c';
[flag,H,theta]=cpopov(A,b,c,mu);
if flag==1
    disp('Problem is feasible');
    H
    disp(sprintf('theta = %f',theta));
else
    disp('Problem is not feasible');
end
```

2.7.3. Оценка запаса устойчивости линейной системы

Пусть имеется матрица A , устойчивая по Гурвицу (т.е. все ее собственные значения лежат строго слева от мнимой оси). Оценим максимально возможное число $\lambda > 0$ такое, что матрица $A + \lambda I$ устойчива. Очевидно точная верхняя грань λ равна модулю вещественной части собственного числа матрицы A , ближайшего к мнимой оси. Поэтому, чтобы определить λ , достаточно использовать стандартную функцию `eig` системы MATLAB, вычисляющую список собственных значений матрицы. В качестве упражнения получим λ с помощью функции `gevpr`.

Исходная задача сводится к следующей оптимизационной задаче:

$$\begin{aligned} \lambda &\rightarrow \max, \\ H(A + \lambda I) + (A' + \lambda I)H &< 0, \quad H > 0, \end{aligned}$$

где H — переменная симметричная матрица. Поскольку функция `gevpr` находит минимум целевой функции, введем переменную $t = -\lambda$. Тогда получаем:

$$\begin{aligned} t &\rightarrow \min, \\ HA + A'H &< 2tH, \quad H > 0. \end{aligned}$$

Таким образом, мы пришли к постановке задачи, для решения которой можно применить функцию `gevpr`.

Приведем текст соответствующей функции:

```
function [flag,H0,lambda0] = rate(A)
%Decay rate of a quadratic Lyapunov function
```

```

sz=size(A);
flag=0;
H0=[ ];
lambda0=[ ];
setlmis([ ]);
H=lmivar(1,[sz(1) 1]);
Hpos=newlmi;
lmiterm([Hpos 1 1 H],-1,1);
MLMI=newlmi;
lmiterm([MLMI 1 1 H],1,A,'s');
lmiterm([-MLMI 1 1 H],2,1);
lmisys=getlmis;
[tmin,xmin]=gevp(lmisys,1,[0.01 100 1e8 10 1]);
if tmin<0
    flag=1;
    H0=dec2mat(lmisys,xmin,H);
    lambda0=-tmin;
end

```

При описании системы линейных неравенств первым следует описать неравенство $H > 0$, так как оно не содержит неизвестной t . Это неравенство гарантирует условие $B(x) > 0$, необходимое для того, чтобы задача имела решение. В качестве второго неравенства берется неравенство $HA + A'H < 2H$. При вызове функции `gevp` второй входной параметр равен единице, т.е. одно из неравенств (последнее) рассматривается как неравенство, в правую часть которого входит t . В результате получаем ограничение $HA + A'H < 2tH$.

2.7.4. Оценка, связанная с неравенством Ляпунова

Рассмотрим следующую задачу, связанную с неравенством Ляпунова (см. [40]):

$$\begin{aligned} \varepsilon &\rightarrow \max, \\ HA + A'H &< -\varepsilon I, \quad 0 < H < I. \end{aligned}$$

Здесь A — заданная квадратная матрица, H — симметричная матричная переменная, ε — скалярный параметр. Для того, чтобы свести задачу к задаче на минимум, введем новую переменную $\alpha = -\varepsilon$. Полученную задачу можно решить как с помощью `mincx`, так и с помощью `gevp`. В качестве примера приведем решение с помощью `mincx`.

```
function [flag,H0,eps0] = lin(A)
% Lyapunov inequality
sz=size(A);
flag=0;
H0=[ ];
eps0=[ ];
I=eye(sz);
setlmis([ ]);
H=lmivar(1,[sz(1) 1]);
[alpha ndec xdec]=lmivar(1,[1 1]);
Hp=newlmi;
lmiterm([-Hp 1 1 H],1,1);
Hn=newlmi;
lmiterm([Hn 1 1 H],1,1);
lmiterm([-Hn 1 1 0],I);
LI=newlmi;
lmiterm([LI 1 1 H],1,A,'s');
lmiterm([-LI 1 1 alpha],1,I);
lmisys=getlmis;
c=zeros(ndec-1,1);
c(ndec)=1;
[alph0,x0]=mincx(lmisys,c,[0.01 100 1e9 10 1]);
if ~isempty(alph0)
    flag=1;
    H0=dec2mat(lmisys,x0,H);
    eps0=-alph0;
end
```

Глава 3. Пакет LMITOOL

3.1. Общая характеристика пакета

Пакет LMITOOL включен в стандартную поставку системы Scilab, т.е., если вы установили Scilab, пакет LMITOOL сразу готов к работе. В то же время, LMITOOL имеет версию и для системы MATLAB (особенности этой версии пакета посвящен раздел 3.4).

Создателями пакета были Р. Никухах (R. Nikoukhah, INRIA), Ф. Дельбек (F. Delebecque, INRIA) и Л. Эль Гауи (L. El Ghaoui, ENSTA). LMITOOL использует пакет SP (Semidefinite Programming package), разработанный Л. Ванденберге (L. Vandenberghe, University of California, Los Angeles) и С. Бойдом (S. Boyd, Stanford University). Подробную документацию по системе Scilab и ее стандартным пакетам можно найти на сайте <http://www.scilab.org>.

Программы на Scilab проще всего оформлять в виде функций, которые можно хранить в отдельных файлах с расширением `.sci`. Для использования этих функций они должны быть предварительно загружены в среде Scilab. Для загрузки можно воспользоваться командой `getf` или пунктом меню `File/Exec`.

Пакет содержит две функции: `lmisolver` и `lmitool`. Со второй из них можно работать в программном или интерактивном режиме. Пакет служит для решения следующей оптимизационной задачи:

$$\begin{aligned} f(X_1, X_2, \dots, X_M) &\rightarrow \min, \\ G_i(X_1, X_2, \dots, X_M) &= 0, \quad i = 1, 2, \dots, p, \\ H_j(X_1, X_2, \dots, X_M) &\geq 0, \quad j = 1, 2, \dots, q. \end{aligned}$$

Здесь X_1, X_2, \dots, X_M — неизвестные вещественные матрицы (матричные переменные), f — скалярная функция (функция цели), G_i — вещественные матрицы-функции, задающие ограничения типа равенств, H_j — симметричные матрицы-функции, задающие ограничения типа неравенств

(неравенства понимаются в смысле неотрицательной определенности).

Как видно из постановки задачи, в отличие от функций пакета LMI-Lab функции пакета LMITOOL позволяют учитывать ограничения типа равенств. На предварительном этапе из исходной задачи получается оптимизационная задача вида

$$\begin{aligned}\tilde{c}'z &\rightarrow \min, \\ \tilde{F}_0 + z_1\tilde{F}_1 + \dots + z_n\tilde{F}_n &\geq 0, \\ Az = b,\end{aligned}$$

где A , \tilde{F}_i — матричные коэффициенты, b , \tilde{c} — векторы-коэффициенты, z — вектор скалярных переменных. Ограничения типа равенств $Az = b$ преобразуются к виду

$$z = Nx + z_0,$$

где x — вектор независимых скалярных переменных. В итоге получаем задачу только с ограничениями типа неравенств:

$$\begin{aligned}c'x &\rightarrow \min, \\ F_0 + x_1F_1 + \dots + x_mF_m &\geq 0.\end{aligned}\tag{3.1}$$

Для решения этой задачи используется функция `semidef` пакета SP. (Таким образом, LMITOOL можно рассматривать как внешний интерфейс для пакета SP.) SP использует один из вариантов алгоритма одновременного решения прямой и двойственной задачи (метод Нестерова—Тодда). На каждом шаге вычисляются p — значение целевой функции прямой задачи и d — значение целевой функции двойственной задачи. Величины p и d служат соответственно верхней и нижней границей оптимального значения. Напомним, что величина $p - d$ называется разрывом двойственности (duality gap). Процесс решения может прекратиться по одной из следующих причин:

- 1) превышено максимально допустимое число итераций (параметр `maxiters`);
- 2) $p - d \leq \text{abstol}$, где `abstol` — заданный параметр (absolute tolerance — абсолютное допустимое отклонение);
- 3) $(p - d)/d \leq \text{reltol}$, где `reltol` — заданный параметр (relative tolerance — относительное допустимое отклонение).

Более подробное описание алгоритма можно найти в [65].

3.2. Описание функций пакета

3.2.1. Функция lmisolver

Это основная функция пакета. Параметры оптимизационной задачи задаются не в ней, а в некоторой вспомогательной функции с произвольным именем. Заголовок функции `lmisolver`:

```
[XLISTF, OPT] = lmisolver(XLIST0,EVALFUNC,options)
```

Выходной параметр `OPT` и входной параметр `options` не обязательны.

`XLIST0` — список, состоящий из матриц или из списков матриц. Он задает начальные приближения для неизвестных матриц. При этом i -й элемент списка `XLIST0` — начальное приближение матрицы X_i . Имена неизвестных матриц можно не задавать явно.

Если начальные приближения не удовлетворяют ограничениям типа равенств, то они игнорируются. Размеры и последовательность матриц, указанных в списке `XLIST0`, используются для формирования выходного списка `XLISTF`.

`EVALFUNC` — функция, которая пишется пользователем и задает ограничения типа равенств и неравенств, а также функцию цели. Эта функция должна иметь следующий заголовок:

```
[LME,LMI,OBJ] = EVALFUNC(XLIST)
```

Здесь `XLIST` — список, имеющий такую же структуру, что и `XLIST0`, `LME` — список матриц или списков, описывающий ограничения типа равенств, `LMI` — список матриц или списков, описывающий ограничения типа неравенств, `OBJ` — функция цели. Если какой либо из параметров `LME`, `LMI`, `OBJ` не задается, то в описании функции вместо него следует указать пустой список `[]`. В частности, если функция цели не задана, решается только задача разрешимости системы неравенств и равенств (feasibility problem).

`options` — вектор размера 5×1 , содержащий параметры работы функции `semidef` пакета SP: `Mbound`, `abstol`, `nu`, `maxiters`, `reltol`. Точный смысл этих параметров объяснен в руководстве по пакету SP (оценка, связанная с параметром M , абсолютное допустимое отклонение, регулятор скорости сходимости, максимальное число итераций, относительное допустимое отклонение). Рядовому пользователю (не специалисту в области выпуклого программирования) эти параметры лучше явно не задавать, а использовать их значения по умолчанию.

`XLISTF` — список, структура которого совпадает со структурой списка `XLIST0`. Он содержит оптимальные значения матричных переменных.

OPT — скалярная величина, равная оптимальному значению функции цели.

3.2.2. Функция lmitool (неинтерактивный режим)

Эта функция служит для автоматизации процесса описания оптимизационной задачи. В частности, она создает файл с расширением .sci с описанием вспомогательных функций. Функцию lmitool можно вызывать с одним или тремя параметрами, либо вообще без параметров. Заголовок функции:

```
txt=lmitool(probname,varlist,datalist)
```

Здесь:

probname — строка, содержащая имя задачи,

varlist — строка, содержащая имена матричных переменных (разделенные запятыми),

datalist — строка, содержащая имена матриц-коэффициентов (разделенные запятыми),

txt — строка, содержащая сообщение о том, что пользователь должен делать дальше.

В текущей папке функция создает файл с именем, указанным в параметре probname, и с расширением .sci.

Пример. Пусть в качестве текущей установлена папка c:\work. Зададим в командной строке:

```
-->txt=lmitool('popov','H,theta','A,b,c,mu')
```

Появляется сообщение

```
functions saved in C:\work/popov.sci
```

и текст:

```
txt =
! To solve your problem, you need to !
!1- edit file C:\work/popov.sci !
!2- load (and compile) your functions: !
! getf('C:\work/popov.sci') !
!3- Define A,b,c,mu and call popov function: !
! [H,theta]=popov(A,b,c,mu) !
!To check the result, use [LME,LMI,OBJ]= !
! popov_eval(list(H,theta)) !
```

Файл `popov.sci` содержит каркас программы:

```
function [H,theta]=popov(A,b,c,mu)
// Generated by lmitool on
Mbound = 1e3;
abstol = 1e-10;
nu = 10;
maxiters = 100;
reltol = 1e-10;
options=[Mbound,abstol,nu,maxiters,reltol];

/////////////DEFINE INITIAL GUESS AND PRELIMINARY
CALCULATIONS BELOW
H_init=...
theta_init=...
///////////

XLIST0=list(H_init,theta_init)
XLIST=lmisolver(XLIST0,popov_eval,options)
[H,theta]=XLIST(:)

///////////////EVALUATION FUNCTION////////////

function [LME,LMI,OBJ]=popov_eval(XLIST)
[H,theta]=XLIST(:)

///////////////DEFINE LME, LMI and OBJ BELOW
LME=...
LMI=...
OBJ=...

Остается только заменить многоточия на конкретный код.
```

3.2.3. Функция `lmitool` (интерактивный режим)

В этом случае функция вызывается с одним параметром или вообще без параметров:

```
txt=lmitool(file)
txt=lmitool()
lmitool()
lmitool
```

Здесь **file** — имя файла с расширением **.sci**, который будет порожден функцией **lmitool**. Выходной параметр **text** содержит текст сообщения, которое формируется функцией после окончания работы (оно автоматически выводится на экран).

Для создания нового файла функция вызывается без параметров. Для модификации существующего файла (созданного ранее с помощью **lmitool**) имя файла следует указать в качестве параметра.

Пример. В командной строке введем:

```
-->lmitool
```

Появляется окно с описанием постановки задачи. Описание можно проchесть, а окно — закрыть, нажав кнопку **OK**. Появляется новое диалоговое окно, в котором предлагаются ввести:

LMI problem name — имя задачи, точнее имя файла, в котором будет храниться программа (в предыдущем примере — **popov**);

Names of unknown matrices — имена матричных переменных через запятую (в предыдущем примере — **H,theta**);

Names of data matrices — имена матриц-коэффициентов (в предыдущем примере — **A,b,c,mu**).

После этого следует нажать кнопку **Ready** (Готово). Появляется окно редактирования с текстом каркаса программы. Текст можно тут же отредактировать или сделать это позднее, в каком-нибудь внешнем редакторе. Нажмите кнопку **OK**. В следующем окне можно отредактировать имя файла и папки, в которой он будет храниться, например

```
c:\work\popov
```

В случае успешного создания файла появится сообщение того же вида, что и для неинтерактивного варианта функции:

```
! To solve your problem, you need to !
!
!1- load your functions using the command:
!
!   getf('C:\temp\popov.sci')
!
!2- Define A,b,c,mu and call function popov as follows:
!
!   [H,theta]=popov(A,b,c,mu)
!
!       Good luck!
```

```
!
!To check the result, use [LME,LMI,OBJ]=
!          popov_eval(list(H,theta)) !
!
```

Для модификации файла программы можно использовать команду
`-->lmitool('popov.sci')`

Если файл не находится в текущей папке, то следует явно указать маршрут.

3.3. Примеры работы с пакетом

3.3.1. Круговой критерий абсолютной устойчивости

Рассмотрим систему линейных матричных неравенств (2.7), к разрешимости которой сводится проверка кругового критерия абсолютной устойчивости. Перепишем (2.7) в виде, пригодном для применения функции `lmisolver`. Получаем одно ограничение типа равенства

$$H - H' = 0$$

и два ограничения типа неравенства

$$\begin{bmatrix} \mu_1\mu_2cc' - HA - A'H & -\frac{1}{2}(\mu_1 + \mu_2)c - Hb \\ -\frac{1}{2}(\mu_1 + \mu_2)c' - b'H & 1 \end{bmatrix} > 0, \quad H > 0.$$

Опишем функцию `circle_eval` для задания системы ограничений:

```
function [LME,LMI,OBJ] = circle_eval(XLIST)
    H = XLIST(:)
    LME = H-H'
    LMI = list(H, ...
        [mu1*mu2*c*c'-H*A-A'*H, -0.5*(mu1+mu2)*c-H*b;
        -0.5*(mu1+mu2)*c'-b'*H, 1])
    OBJ = []

```

(Пожалуйста, обратите внимание, что при разбиении оператора задания LMI на несколько строк многоточие *следует* ставить после элемента списка `H` и *нельзя* ставить внутри описания матрицы — таковы особенности синтаксиса Scilab.) В первом операторе из списка извлекается матричная переменная (ее размеры определяются при задании начальных условий). В данном случае рассматривается только задача разрешимости, поэтому для целевой функции полагаем `OBJ = []`.

Если ограничений много и они достаточно сложные, то для большей наглядности список лучше создавать более структурно:

```
function [LME,LMI,OBJ] = circle_eval(XLIST)
    H = XLIST(:)
    LME = H-H'
    LMI = list()
    LMI(1) = H
    LMI(2)= [mu1*mu2*c*c' -H*A-A'*H, -0.5*(mu1+mu2)*c-H*b;
              -0.5*(mu1+mu2)*c'-b'*H, 1]
    OBJ = [ ]
```

С помощью следующей функции проверяем разрешимость системы для конкретных числовых значений. Задаем нулевые начальные значения и вызываем функцию `lmisolver`:

```
function H=test()
    mu1=2;
    mu2=3;
    s=poly(0,'s');
    numerator=3*s^4+2.5*s^3+s^2+0.8*s+0.1;
    denominator=s^5 -1.7*s^4+0.57*s^3+0.21*s^2-0.07*s-0.01;
    w=numerator/denominator;
    sys=tf2ss(w);
    A=sys.A;
    b=sys.B;
    c=-sys.C';
    H_init = zeros(A);
    XLIST0=list(H_init);
    [XLIST,OPT]=lmisolver(XLIST0,circle_eval);
    [H]=XLIST(:);
```

Функция `tf2ss` служит для получения коэффициентов уравнений в пространстве состояний по передаточной функции системы.

Поместим обе функции в файл с расширением `.sci` (имя файла значения не имеет). Пусть, для определенности, описание обеих функций находится в файле с именем `circle.sci`. Загрузим этот файл в среду Scilab, например:

```
-->getf('C:\work\circle.sci')
(естественно, маршрут зависит от того, где конкретно расположен файл).
Теперь можно вызвать функцию test из командной строки (или из файла сценария):
```

```
-->H=test()
```

Для данного числового примера, на экран будут выведены следующие сведения:

```
Construction of canonical representation
Basis Construction
```

```
FEASIBILITY PHASE.
```

| primal obj. | dual obj. | dual. gap |
|-------------|------------|-----------|
| 4.32e-002 | -5.84e+002 | 5.84e+002 |
| 3.68e+001 | -5.84e+002 | 6.21e+002 |
| 3.88e+001 | -3.82e+001 | 7.70e+001 |
| 1.19e+001 | -2.21e+001 | 3.40e+001 |
| 6.37e+000 | -1.87e+000 | 8.24e+000 |
| 4.98e-001 | -9.99e-001 | 1.50e+000 |
| 1.12e-001 | -2.27e-001 | 3.40e-001 |
| 2.38e-002 | -8.38e-002 | 1.08e-001 |
| 1.09e-002 | -2.19e-002 | 3.28e-002 |
| 4.69e-003 | -1.11e-002 | 1.58e-002 |
| 2.20e-003 | -4.70e-003 | 6.90e-003 |
| 8.01e-004 | -2.32e-003 | 3.12e-003 |
| -7.43e-004 | -1.55e-003 | 8.11e-004 |

| primal obj. | dual obj. | dual. gap |
|-------------|------------|-----------|
| 4.32e-002 | -5.84e+002 | 5.84e+002 |
| 3.68e+001 | -5.84e+002 | 6.21e+002 |
| 3.88e+001 | -3.82e+001 | 7.70e+001 |
| 1.19e+001 | -2.21e+001 | 3.40e+001 |
| 6.37e+000 | -1.87e+000 | 8.24e+000 |
| 4.98e-001 | -9.99e-001 | 1.50e+000 |
| 1.12e-001 | -2.27e-001 | 3.40e-001 |
| 2.38e-002 | -8.38e-002 | 1.08e-001 |
| 1.09e-002 | -2.19e-002 | 3.28e-002 |
| 4.69e-003 | -1.11e-002 | 1.58e-002 |
| 2.20e-003 | -4.70e-003 | 6.90e-003 |
| 8.01e-004 | -2.32e-003 | 3.12e-003 |
| -7.43e-004 | -1.55e-003 | 8.11e-004 |

```
Target value reached
feasible solution found
H =
! 9.4739537  2.8309747 -4.5712564 - .8691594  34.03019 !
! 2.8309747  1.2011726 -1.4163251 - .0432901  8.0344382 !
! -4.5712564 -1.4163251  6.4349141  2.1831707 -60.238753 !
! - .8691594 - .0432901  2.1831707  1.193925 -24.189243 !
! 34.03019    8.0344382 -60.238753 -24.189243  629.08564 !
-->
```

Как видно из приведенного текста, для каждого шага алгоритма приводятся значения целевых функций прямой и двойственной задачи, а также величина разрыва двойственности. Вычисления прекращаются, когда разрыв двойственности становится достаточно малым.

Описанный порядок работы применим для всех версий Scilab. В версии 3.0 функция `getf` допустима, но считается устаревшей. Для обеих

функций программы добавим в конец их описаний по строке:

```
endfunction
```

Тогда вместо вызова функции `getf` можно воспользоваться пунктом меню `File/Exec...`. В случае успешной компиляции функций будет выведено сообщение:

```
exec done
```

Дальнейший вызов функций возможен из командной строки или из сценария (для выполнения сценария можно использовать тот же пункт меню `Exec`).

3.3.2. Критерий Попова

Как отмечалось в предыдущей главе, проверка критерия Попова сводится к разрешимости системы

$$H - H' = 0,$$

$$\begin{bmatrix} -HA - A'H & -\frac{1}{2}\mu c - Hb - \frac{1}{2}\theta A'c \\ -\frac{1}{2}\mu c' - b'H - \frac{1}{2}\theta c'A & 1 - \theta c'b \end{bmatrix} > 0, \quad H > 0$$

относительно неизвестных матрицы H и числа θ . Соответствующие функции могут иметь вид:

```
function [H,theta]=test()
    s=poly(0,'s');
    mu=4;
    numerator=-s+1;
    denominator=s^2 +5*s+6;
    w=numerator/denominator;
    sys=tf2ss(w);
    A=sys.A;
    b=sys.B;
    c=-sys.C';
    H_init = zeros(A);
    theta_init=0;
    XLIST0=list(H_init,theta_init);
    XLIST=lmisolver(XLIST0,popov_eval);
    [H,theta]=XLIST(:);
```

```

function [LME,LMI,OBJ] = popov_eval(XLIST)
[H,theta] = XLIST(:)
LME = H-H'
LMI = list()
LMI(1) = H
LMI(2)= [-H*A-A'*H, -0.5*mu*c-H*b-0.5*theta*A'*c;
           -0.5*mu*c'-b'*H-0.5*theta*c'*A, 1-theta*c'*b]
OBJ = []

```

3.3.3. Проверка условий пассивности

Пусть заданы коэффициенты: A — квадратная $n \times n$ -матрица, b , c — $n \times m$ -матрицы. Требуется найти симметричную $n \times n$ -матрицу H , удовлетворяющую системе ограничений [6]:

$$HA + A'H \leq 0, \quad Hb = c, \quad H > 0. \quad (3.2)$$

Заметим, что пакет LMILab в этой ситуации неприменим.

К соотношениям (3.2) сводится проверка условий пассивности линейной системы [35]. Пусть система управления описывается уравнениями

$$\frac{dx}{dt} = Ax + bu, \quad y = c'x, \quad (3.3)$$

причем вектора входа u и выхода y имеют одинаковые размерности. Система (3.3) называется пассивной, если для всех чисел $T > 0$, всех входов $u(t)$ и для всех решений с нулевыми начальными условиями $x(0) = 0$ вход и выход системы связаны неравенством

$$\int_0^T u(t)'y(t) dt \geq 0.$$

Пусть матрица H удовлетворяет соотношениям (3.2). Тогда для всех векторов x , u справедливо неравенство

$$2x'H(Ax + bu) - 2(c'x)'u \leq 0.$$

Рассмотрим положительно определенную квадратичную форму $V(x) = x'Hx$. Ее производная в силу системы (3.3) удовлетворяет неравенству

$$\frac{dV}{dt} + 2u'y \leq 0,$$

из которого очевидно вытекает свойство пассивности.

Для отладки возьмем

$$A = \begin{bmatrix} 0 & 1 \\ -6 & -5 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad c = \begin{bmatrix} 6 \\ 1 \end{bmatrix}.$$

Задача имеет решение, например, при

$$H = \begin{bmatrix} 6 & 0 \\ 0 & 1 \end{bmatrix}.$$

Приведем текст программы:

```
function [H,theta]=test()
A=[0 1; -6 -5]
b=[1; 1]
c=[6; 1]
H_init = eye(A)
XLIST0=list(H_init)
XLIST=lmisolver(XLIST0,lyap_eval)
[H]=XLIST(:)

function [LME,LMI,OBJ] = lyap_eval(XLIST)
[H] = XLIST(:)
LME = list()
LME(1) = H-H'
LME(2) = H*b-c
LMI = list()
LMI(1) = H
LMI(2)= -H*A-A'*H
OBJ = [ ]
```

При вызове в командной строке

```
-->H=test()
```

получаем сообщение:

```
Construction of canonical representation
Basis Construction
recomputing initial guess
```

```

FEASIBILITY PHASE.

primal obj.    dual obj.    dual. gap

2.20e+000    -4.12e+000    6.32e+000
-1.24e-001    -1.64e+000    1.52e+000

Target value reached
feasible solution found
H =
!
!      5.2374557      .7625443 !
!      .7625443      .2374557 !

```

Поскольку заданное начальное значение заведомо не удовлетворяло ограничению типа равенства, оно не учитывалось.

3.3.4. Классическая задача линейного программирования

Рассмотрим оптимизационную задачу [55]:

$$e'x \rightarrow \min,$$

$$Ax + b \geq 0,$$

$$Cx + d = 0,$$

где A, C — матрицы, b, d, e — векторы.

Вспомогательная функция в этом случае имеет вид:

```

function [LME,LMI,OBJ]=linprog_eval(XLIST)
x=XLIST(:)
[m,n]=size(A)
LME=C*x+d
LMI=list()
tmp=A*x+b
for i=1:m
    LMI(i)=tmp(i)
end
OBJ=e'*x

```

3.4. Пакет LMITOOL в системе MATLAB

Вариант пакета LMITOOL для системы MATLAB не связан с проектом Scilab и распространяется как отдельный программный продукт.

В 1996 г. была выпущена версия LMITOOL-2.1 (модифицирована в 1997 и 1999 г.г.), которая допускала работу с тремя пакетами для решения задач полуопределенного программирования — SP, SDPack и SDPHA, причем эти пакеты были специально модифицированы для взаимодействия с LMITOOL. Сам пакет LMITOOL также подвергся некоторой переделке — в функции `lmisolver` и `lmitool` были добавлены дополнительные параметры. Дистрибутив версии 2.1 был выложен на сайте ENSTA, но, начиная с 2003 г., доступ к нему прекращен.

В сентябре 2001 г. была выпущена версия LMITOOL-2.2. Она может быть загружена с сайта одного из разработчиков, Лорана Эль Гауи: <http://robotics.eecs.berkeley.edu/~elghaoui/>.

В отличие от LMITOOL-2.1, версия 2.2 работает только с решателем SeDuMi. К сожалению, это не единственное различие двух версий. Старая версия была достаточно хорошо отлажена и функционировала устойчиво, в то время как новая содержит множество мелких недоработок. Судя по тому, что с 2001 г. не было сделано никаких исправлений и модификаций, разработчики прекратили активную поддержку данного программного продукта. Эти обстоятельства не позволяют рекомендовать LMITOOL-2.2 для широкого круга пользователей, тем более, что для интерфейса с решателем SeDuMi в системе MATLAB могут быть использованы такие пакеты как SeDuMi Interface и YALMIP.

Глава 4. Пакет SeDuMi Interface

4.1. Общая характеристика пакета

Пакет SeDuMi Interface относится к чисто интерфейсным пакетам и ориентирован на работу с решателем SeDuMi (Self-Dual Minimization), что и подчеркнуто в его названии. Этот программный продукт, как и сам решатель SeDuMi, является бесплатным и находится в свободном доступе в Интернете. Приведем еще раз адреса, по которым можно найти дистрибутивы SeDuMi Interface и SeDuMi:

<http://www.laas.fr/peaucell/SeDuMiInt.html>,
<http://fewcal.kub.nl/sturm/software/sedumi.html>.

Вместе с пакетом SeDuMi Interface пользователь получает достаточно подробное его описание в файлах `sdmguide.pdf` или `sdmguide.ps`, а инструкцию по установке решателя SeDuMi можно найти в файле `Install_dos.htm`. Для установки пакета в среде MATLAB необходимо создать две папки для приложений, например:

`c:\Mathlab\toolbox\SeDuMiInt104` — для интерфейсного пакета,
`c:\Mathlab\toolbox\SeDuMi105` — для решателя.

Далее разархивировать в эти папки файлы пакетов. Затем нужно установить в системе MATLAB пути к папкам обоих пакетов, включая вложенные папки (см. приложение 1). После этого SeDuMi готов к работе.

Как и пакет LMILab, пакет SeDuMi Interface создан для использования в среде MATLAB. Для версий SeDuMi 1.05 и SeDuMi Interface 1.04, на примере которых и будут разбираться особенности пакета, потребуется MATLAB версии 5.0 или выше.

На первый взгляд SeDuMi Interface является всего лишь еще одним видом интерфейса, наряду с описанными ранее пакетами LMILab и LMITOOL. Однако он и его решатель SeDuMi имеют несколько существенных преимуществ [56].

- В настоящее время (2004 г.) решатель SeDuMi считается одним из

наиболее эффективных для средних по объему задач. Об этом свидетельствует и тот факт, что все разработанные в последние годы интерфейсы предусматривают работу с SeDuMi.

- Матричные переменные и входные параметры могут иметь комплексные значения.
- Пакет обладает достаточно большим выбором встроенных типов данных для описания матричных переменных задачи. Если в пакете LMILab предусмотрены только три основных типа матриц (симметричные блочно-диагональные, прямоугольные произвольной и заданной структуры), то SeDuMi Interface сверх того позволяет автоматически описывать такие типы матриц, как антисимметричные, эрмитовы и антиэрмитовы.
- Решатель SeDuMi позволяет обрабатывать задачи, содержащие как ограничения в виде неравенств, так и ограничения-равенства.
- Линейная целевая функция может включать в себя оператор вычисления следа `trace`.
- При описании матричных ограничений может использоваться прямое произведение Кронекера.

Таким образом, круг оптимизационных задач, которые могут быть исследованы или решены с помощью SeDuMi Interface, значительно расширяется по сравнению с другими пакетами.

В отличие от рассмотренных ранее пакетов LMILab и LMITOOL, которые представляют собой пакеты функций, пакет SeDuMi Interface использует средства объектно-ориентированного программирования. В SeDuMi Interface определен единственный класс (объектный тип данных) с именем `sdpb` (semi-definite program problem), и большинство функций пакета описаны как методы этого класса. Кроме того, имеется несколько внешних функций (не связанных с каким либо классом). Впрочем, для пользователя это обстоятельство не слишком существенно, так как в языке программирования MATLAB синтаксические различия между вызовом внешней функции и вызовом метода класса почти не проявляются.¹

Разработчики SeDuMi Interface не предусмотрели возможности интерактивного режима создания оптимизационной задачи. Отчасти это

¹При вызове метода имя объекта указывается в качестве одного из фактических параметров.

объясняется принципиальными соображениями, т.к. использование графического интерфейса в пакетах LMILab и LMITOOL замедляет работу программы, в результате чего преобразование задачи часто занимает больше времени, чем ее решение [56].

Функции, входящие в пакет, поделены на три группы: функции, создающие новую оптимизационную задачу, а также модифицирующие уже существующую; функция, вызывающая программу-решатель; вспомогательные функции, предназначенные для получения информации о задаче, а также демонстрации возможностей пакета.

4.2. Создание оптимизационной задачи

Создание оптимизационной задачи распадается на несколько этапов: описание матричных переменных, описание ограничений-неравенств, описание ограничений-равенств.

Вся информация о решаемой оптимизационной задаче содержится в переменной, которая является объектом класса `sdmpb`. Эта переменная включает в себя информацию о матричных переменных задачи, об ограничениях (матричных равенствах и неравенствах), о целевой функции и собственно оптимальное решение. При этом сначала создается «пустой» объект задачи, с неинициализированными полями данных, а затем эти поля заполняются.

4.2.1. Функция `sdmpb`

Назначением функции `sdmpb` является создание нового, «пустого» объекта класса `sdmpb`.² Вызов функции имеет следующий вид

```
pbname = sdmpb(label)
```

Здесь `pbname` — имя создаваемой переменной класса `sdmpb`, с помощью которой будут происходить все обращения к задаче, `label` — необязательный параметр, описывающий назначение переменной и содержащий имя задачи в виде строки.

Пример.

```
>> LMCpb=sdmpb('My Problem')
```

В результате получим сообщение:³

²Говоря более точно, она служит конструктором этого класса. В языке программирования MATLAB имя конструктора всегда совпадает с именем класса.

³Заметим, что сокращение LMC, которое неоднократно встречается в документации, означает Linear Matrix Constraint — линейное матричное ограничение.

```
LMC problem: My Problem
```

```
no matrix variable
no equality constraint
no inequality constraint
no linear objective
unsolved
```

В первой строке сообщения — имя оптимизационной задачи, и далее построчно — содержимое пяти полей объекта `LMCpb`: матричные переменные, ограничения типа равенств, ограничения типа неравенств, линейная целевая функция и, наконец, оптимальное решение. Поскольку `LMCpb` был только создан, но не инициализирован, в примере все эти поля пустые. Их необходимо последовательно заполнить.

4.2.2. Функция `sdmvar`

Функция `sdmvar` предназначена для определения новой матричной переменной создаваемой оптимизационной задачи. Отметим, что все матричные переменные рассматриваемого пакета могут быть как вещественными, так и комплексными. Возможность решения задач, включающих комплексные матричные переменные, появилась в пакете SeDuMi Interface начиная с версии 1.02 и она выгодно отличает этот пакет от других (например, LMI Lab). Заголовок этой функции имеет следующий вид:

```
[pbname, Xindex] = sdmvar(pbname, nbrow, nbcoll, name)
```

Здесь `pbname` — имя объекта класса `sdmpb`, для которого объявляется новая матричная переменная. Этот объект упоминают дважды — в списках входных и выходных параметров, так как в результате создания новой матричной переменной он изменяется.⁴ Параметры `nbrow`, `nbcoll` в простейшем случае равны числу строк и столбцов матричной переменной (общая схема всех возможных значений этих параметров будет рассмотрена ниже); `name` — необязательный параметр-строка, содержащий имя матричной переменной (значением по умолчанию является `name='NO NAME'`); `Xindex` — порядковый номер создаваемой матричной переменной в списке всех матричных переменных задачи `pbname`. Переменные numеруются последовательно, по мере их создания, т.е., если уже объявлена $k - 1$ матричная переменная, то `Xindex` принимает значение k . Отметим, что входной параметр `name` несет только «эстетическую» нагрузку

⁴ Применение подобной схемы вызвано тем, что в языке MATLAB отсутствует возможность передачи параметров по адресу или ссылке.

Таблица 4.1.

| <code>nbcoll</code> | <code>nbrow = n</code> | <code>nbrow = n*i</code> |
|---------------------|---|--|
| пропущен | вещественная $n \times n$ -матрица произвольного вида | комплексная $n \times n$ -матрица произвольного вида |
| 'd' | вещественная диагональная $n \times n$ -матрица | комплексная диагональная $n \times n$ -матрица |
| 's' | вещественная симметричная $n \times n$ -матрица | комплексная симметричная $n \times n$ -матрица |
| 'h' | не определена | эрмитова $n \times n$ -матрица |
| 'as' | вещественная антисимметричная $n \times n$ -матрица | комплексная антисимметричная $n \times n$ -матрица |
| 'ah' | не определена | антиэрмитова $n \times n$ -матрица |

и предназначен для более наглядного вывода на экран информации о задаче. При вызове других функций используется не имя переменной, а ее порядковый номер (индекс). По умолчанию все матричные переменные имеют элементы вещественного типа. Если при вызове функции `sdimvar` указаны только два первых параметра, то параметры `nbcoll` и `name` получают значения `nbcoll=nbrow` и `name='NO NAME'` (квадратная матрица без имени).

Для наглядности мы свели значения входных параметров `nbrow` и `nbcoll`, которые используются при описании квадратных матриц, в таблицу 4.1. В крайнем левом столбце таблицы перечислены возможные значения параметра `nbcoll`, в верхней строке таблицы перечислены значения параметра `nbrow`, а в остальных ячейках таблицы указаны типы матриц, которые могут быть описаны соответствующей комбинацией `nbrow` и `nbcoll`. Здесь n — целое положительное число, i — мнимая единица ($i = \sqrt{-1}$).

Отдельно рассмотрим ситуацию, когда матричная переменная `name` составляется из уже существующих скалярных переменных (decision variables). В этом случае параметр `nbcoll` равен строке 'st', что означает создание прямоугольной матрицы с особой структурой (structured rectangular), а `nbrow` сам становится матрицей, размеры которой совпадают с размерами создаваемой матрицы. Способ формирования матрицы `nbrow` описан в табл. 4.2. Первый столбец содержит возможные значения (k, j) -го элемента матричного параметра `nbrow`; здесь n — порядковый номер (индекс) скалярной переменной. Во втором столбце указан характер зависимости от этой скалярной переменной (k, j) -го элемента матричной переменной X .

Таблица 4.2.

| $\text{nbrow}(k, j)$ | $X(k, j)$ |
|----------------------|--|
| 0 | 0 |
| $+n$ | n -я вещественная скалярная переменная |
| $-n$ | $(-1) * n$ -ю вещественную скалярную переменную |
| $+n + n*i$ | n -я комплексная скалярная переменная |
| $-n - n*i$ | $(-1) * n$ -ю комплексную скалярную переменную |
| $+n - n*i$ | комплексное сопряжение n -ой скалярной переменной |
| $-n + n*i$ | $(-1)*$ комплексно-сопряженную n -ю скалярную переменную |

Примеры. Рассмотрим задачу LMCpb (см. пример к разделу 4.2.1), и определим для нее матричные переменные различной структуры.

1. Скалярная переменная с именем eps.

```
[LMCpb, eps_index]=sdmvar(LMCpb, 1, 1, 'eps');
```

2. Прямоугольная вещественная $m \times n$ -матрица Y .

```
m=3; n=4;
[LMCpb, Y_index]=sdmvar(LMCpb, m, n, 'Y');
```

3. Квадратная антисимметричная матрица A .

```
[LMCpb, A_index]=sdmvar(LMCpb, m, 'as', 'A');
```

Выведем теперь на экран состояние задачи LMCpb :

```
>>LMCpb
LMC problem: MyLMC
matrix variables:      index          name
                  1              eps
                  2              Y
                  3              A
```

```

no equality constraint
no inequality constraint
no linear objective
unsolved

```

В SeDuMi Interface предусмотрена возможность получения полной или частичной информации о задаче. Выведем на экран информацию о структуре объявленных переменных. Для этого используем функцию `get`, где первый аргумент — имя объекта задачи, второй аргумент — строка '`vardec`', означающая, что на экран будет выведена именно структура матричной переменной, а третий аргумент — индекс той матричной переменной, чью структуру необходимо определить. Выходным параметром функции `get` является матрица, отражающая структуру переменной (более подробно функция `get` будет рассмотрена в разделе 4.5).

```

>> eps_struct=get(LMCpb,'vardec',eps_index)
eps_struct =
    1
>> Y_struct=get(LMCpb,'vardec',Y_index)
Y_struct =
    2      5      8     11
    3      6      9     12
    4      7     10     13
>> A_struct=get(LMCpb,'vardec',A_index)
A_struct =
    0     -14     -15
   14      0     -16
   15     16      0

```

Содержимое матриц `eps_struct`, `Y_struct`, `A_struct` показывает, каким образом система перенумеровала элементы матричных переменных. Например, если обозначить вектор независимых переменных через x , то антисимметричная квадратная матричная переменная A определяется через этот вектор следующим образом:

$$A = \begin{bmatrix} 0 & -x_{14} & -x_{15} \\ x_{14} & 0 & -x_{16} \\ x_{15} & x_{16} & 0 \end{bmatrix}.$$

Добавим теперь к задаче `LMCpb` матричные переменные более сложной структуры.

4. Комплексная эрмитова матрица H .

```
[LMCpb,H_index]=sdmvar(LMCpb,3*i,'h','H');
```

Выведем информацию об этой переменной:

```
>> H_struct=get(LMCpb,'vardec',H_index);
>> num2str(H_struct)
H_struct =
    17+0i    18-18i   19-19i
    18+18i    20+0i    21-21i
    19+19i    21+21i   22+0i
```

Поясним, что функция `num2str` — стандартная функция MATLAB (а не рассматриваемого пакета). Она использована в примере потому, что для комплексной матрицы функция `get` выводит матрицу структуры в формате вещественных чисел, что загромождает экран.

5. Матрица S симметрична, имеет размеры 4×4 и структуру

$$S = \begin{bmatrix} x_{23} & 0 & 0 & 0 \\ 0 & x_{23} & 0 & 0 \\ 0 & 0 & x_{24} & x_{25} \\ 0 & 0 & x_{25} & x_{26} \end{bmatrix}.$$

Данная матрица может быть описана с помощью следующей последовательности команд:

```
decnb=get(LMCpb,'vardecnb');
nbrow=[decnb+1 0 0 0;
        0 decnb+1 0 0;
        0 0 decnb+2 decnb+3;
        0 0 decnb+3 decnb+4];
[LMCpb,S_index]=sdmvar(LMCpb,nbrow,'st','S');
```

Мы считаем, что матрица S не зависит от ранее описанных скалярных переменных. Поэтому, чтобы задать нужным образом массив `nbrow`, необходимо знать число уже созданных скалярных переменных `decnb`. Это можно сделать с помощью функции `get`, указав строку '`'vardecnb'`' вторым аргументом. Далее вручную приходится создавать матрицу индексов `nbrow`. Выведем матрицу структуры переменной S в явном виде:

```
>> S_struct=get(LMCpb,'vardec',S_index)

S_structure =
 23    0    0    0
```

$$\begin{array}{cccc} 0 & 23 & 0 & 0 \\ 0 & 0 & 24 & 25 \\ 0 & 0 & 25 & 26 \end{array}$$

Заметим, что при создании матричной переменной заданной структуры функция `sdmvar` уступает в удобстве использования аналогичной функции `lmivar` пакета LMILab. Для блочной матричной переменной фактически вручную приходится задавать ее структуру, в то время как функция `lmivar` обладает более гибким механизмом описания таких матриц.

6. Прямоугольная матрица U полностью зависит от уже существующих скалярных переменных.

Составим матрицу U из двух блоков

$$U = [\bar{H} \quad A],$$

где H и A — описанные ранее матричные переменные (см. примеры 3 и 4), черта сверху обозначает комплексное сопряжение. Тогда для создания матричной переменной U необходимо выполнить следующие операторы:

```
U_nbrow=[conj(H_struct), A_struct];
[LMCpb,U_index]=sdmvar(LMCpb,U_nbrow,'st','U');
```

4.2.3. Задание матричных ограничений

Вторым шагом в создании оптимизационной задачи является задание матричных ограничений. Рассматриваемые задачи могут включать в себя ограничения как в виде матричных равенств, так и в виде неравенств.

В SeDuMi Interface ограничение создается в два этапа. Сначала определяется «пустое» ограничение определенной структуры, затем оно наливается содержанием. Поэтому для каждого вида ограничений в пакете присутствует пара функций. Для ограничений типа неравенств — это функции `sdmlmi` и `sdmineq`, а для ограничений типа равенств — функции `sdmlme` и `sdmeq`. Рассмотрим эти функции подробно.

Функция `sdmlmi`

Функция `sdmlmi` позволяет создать «пустое» ограничение типа линейного матричного неравенства. В пакете SeDuMi Interface все матричные неравенства являются «левосторонними» и нестрогими, вида

$$L(x) \leq 0,$$

где $L(x)$ — эрмитова (в вещественном случае — симметричная) матрица. Поэтому для описания такого неравенства необходимо указать только порядок соответствующей эрмитовой матрицы $L(x)$ и, при желании, определить имя неравенства. В общем виде вызов функции `sdmlmi` будет выглядит так:

```
[pbname,lmiindex] = sdmlmi(pbname,dim,name)
```

Здесь, как и раньше, `pbname` — имя объекта той задачи, к которой добавляется ограничение. Параметр `dim` определяет размеры эрмитовой матрицы, которая стоит в левой части матричного неравенства. При создании матрицы простой структуры `dim` — это ее порядок, если же создается неравенство блочной структуры, тогда `dim` — это массив размеров блоков, расположенных на главной диагонали матрицы. Входной параметр `name` — необязательный параметр-строка, содержащий текстовое описание неравенства. К выходным параметрам относится имя обновленного объекта `pbname` и индекс объявленного линейного матричного неравенства `lmiindex`. Именно этот индекс и понадобится для дальнейшей работы.

Пример. Пусть с помощью матрицы $L(x)$ размера 3×3 необходимо создать ограничение, не имеющее блочной структуры. Дадим ему имя '`H>0`'. Это можно сделать следующим образом:

```
[LMCpb,Ineq1_index] = sdmlmi(LMCpb,3,'H>0');
```

В результате при получении информации о текущем состоянии задачи `LMCpb` поле, отвечающее за ограничение в виде неравенств, приобретет значение:

| inequality constraints: | index | meig | name |
|-------------------------|-------|------|------|
| | 1 | -Inf | H>0 |

Столбцы `index` и `name` соответственно содержат индекс неравенства и его описание, в столбец `meig` выводится минимальное собственное значение матрицы $L(x)$. При создании неравенства это поле по умолчанию принимает значение $-\infty$.

Надо отметить, что к дистрибутиву SeDuMi Interface прилагается достаточно полное описание функций с примерами, однако по какой-то причине отсутствует описание функции `sdmlmi` в случае объявления матричного неравенства блочной структуры. Поэтому рассмотрим еще один пример и для сравнения возьмем немного измененный пример 2 раздела 2.1.4.

Пример. Создадим структуру матричного неравенства, на основе которого потом можно будет создать ограничение вида

$$\begin{bmatrix} A'HA - H & HA'b \\ b'HA & b'Hb \end{bmatrix} < F,$$

где H — матричная переменная размера $m \times m$, A и F — заданные постоянные матрицы подходящего размера, b — заданный вектор-столбец. Тогда для задания неравенства должна быть вызвана функция

```
[LMCpb, Ineq2_index]
    = sdmlmi(LMCpb, [m 1], 'LMI 2');
```

В результате в задаче LMCpb описаны два неравенства:

```
inequality constraints:
    index      meig      name
        1        -Inf      H>0
        2        -Inf      LMI 2
```

Функция `sdmineq`

Еще раз отметим, что в пакете SeDuMi Interface задание структуры матричного неравенства и определение содержимого его блоков выполняются двумя различными функциями (`sdmlmi` и `sdmineq`), в то время как в пакете LMILab, к примеру, эти действия выполняются одной и той же функцией `lmiterm`.

Функция `sdmineq` предназначена для добавления некоторого члена в матричное неравенство, уже созданное с помощью `sdmlmi`. В общем виде заголовок этой функции выглядит так:

```
pbname = sdmineq(pbname, [lmiindex blrow blcol], Xindex, L, R)
```

Здесь `lmiindex` — индекс неравенства задачи `pbname`, которое будет модифицировано. Если последний аргумент `R` не пуст, то в неравенстве `lmiindex` к блоку с позицией (`blrow, blcol`) добавляется слагаемое вида LXR (X — матричная переменная, которой соответствует индекс `Xindex`) и одновременно к симметричному блоку с позицией (`blcol, blrow`) добавляется эрмитово сопряженное слагаемое $(LXR)^*$. (Звездочка обозначает эрмитово сопряжение, т.е. транспонирование и комплексное сопряжение.) Если же параметр `R=[]` или пропущен, то `blrow=blcol` и функция `sdmineq` добавляет слагаемое вида LXL^* к диагональному блоку. Остается лишь при использовании этой функции принять во внимание несколько ее особенностей.

1. Как уже было отмечено, в SeDuMi Interface новые члены можно добавить только в левую часть неравенства. Если в исходном неравенстве блочный член находился справа от знака $<$, то его следует умножить на -1 , например, так:

```
pbname = sdmneq(pbname,[lmiindex blrow blcol],Xindex,-L,R);
```

Тот же самый результат будет достигнут и командой

```
pbname = sdmneq(pbname,[-lmiindex blrow blcol],Xindex,L,R);
```

(т.е. индекс неравенства указывается со знаком минус).

2. Для добавления постоянного слагаемого вида LR , не содержащего переменных задачи, входной параметр $Xindex$ полагают равным нулю:

```
pbname = sdmneq(pbname,[lmiindex blrow blcol],0,L,R);
```

3. Все матричные неравенства в пакете SeDuMi Interface представлены эрмитовыми матрицами. Именно поэтому при добавлении члена к блоку (i, j) , система автоматически добавляет эрмитово сопряженный член к блоку (j, i) . Тем не менее, при попытке вручную добавить блок, симметричный заданному, сообщение об ошибке не выдается (как это, например, происходит в пакете LMILab) и, следовательно, контроль такой ситуации полностью лежит на пользователе. Это свойство пакета особенно важно учитывать при работе с диагональными блоками.

Пример. Возьмем матричное неравенство, структура которого определена в предыдущем примере:

$$\begin{bmatrix} A'HA - H & A'Hb \\ b'HA & b'Hb \end{bmatrix} < F,$$

и добавим в диагональный блок $(2, 2)$ член, равный $b'Hb$. Это можно сделать так:

```
LMCpb=sdmneq(LMCpb,[Ineq2_index 2 2], H_index, b', 0.5*b);
```

Поскольку мы явно указали значение пятого аргумента $R = 0.5b$, функция `sdmneq` добавит к матричному блоку два слагаемых вида $0.5b'Hb$, которые в сумме дадут необходимый член $b'Hb$. То же самое можно сделать и более простым способом:

```
LMCpb=sdmneq(LMCpb,[Ineq2_index 2 2], index_H, b');
```

Мы опустили последний параметр R , поэтому функция `sdmneq` добавит в неравенство только одно слагаемое вида $b'Hb$.

Теперь добавим в правую часть неравенства постоянную матрицу F :

```
LMCpb=sdmineq(LMCpb,-Ineq2_index,0,1,0.5*F);
```

Слагаемое добавляется справа от знака $<$, поэтому перед индексом неравенства появляется минус. В результате к правой части прибавляются два слагаемых $1 \cdot \frac{1}{2}F$ и $\frac{1}{2}F^* \cdot 1$, а поскольку матрица F является эрмитовой, то при сложении достигается необходимый нам результат. Отметим типичную ошибку: попытка добавить матрицу F с помощью более простой команды

```
LMCpb=sdmineq(LMCpb,-Ineq2_index,0,F);
```

приведет к тому, что будет добавлена матрица FF^* .

Пример. Похожая на рассмотренный пример ситуация возникает, если надо задать неравенство, в которое входит матричная переменная без множителей ($L = R = 1$). Потребуем положительную определенности матричной переменной H , т.е. определим неравенство $H > 0$. Приведенное ограничение можно задать двумя способами. В первом из них явно указываются левый и правый множители, при этом аналогично предыдущему примеру:

```
LMCpb=sdmineq(LMCpb, Ineq1_index, H_index, -0.5, 1);
```

Заметим, что для получения правильного знака неравенства мы на этот раз домножили на -1 один из коэффициентов.

Второй способ заключается в том, что опускаются два последних входных параметра функции `sdmineq`. Тогда система принимает их равными единичным матрицам и при этом к диагональным блокам не добавляется транспонированное комплексно-сопряженное слагаемое. В таком случае команда, задающая неравенство, будет следующей:

```
>>LMCpb=sdmineq(LMCpb, -Ineq1_index, H_index);
```

Очевидно, что если пропущен один или оба последних параметра, то должны быть выполнены следующие условия: `nbrow=nbcol` и переменная с номером `Xindex` должна быть эрмитовой матрицей. В противном случае `sdmineq` выдает предупреждение.

4. В задачах оптимального управления встречается ситуация, когда один из членов неравенства может зависеть не от самой матричной переменной X и не от эрмитово сопряженной матрицы X^* , а от транспонированной матрицы X' . Средства решателя SeDuMi позволяют обрабатывать и такой случай тоже.⁵

⁵ В математических выражениях мы различаем транспонирование и эрмитово сопряжение, обозначая их разными символами (соответственно штрих и звездочка). В

Для того, чтобы добавить в некоторый блок неравенства член вида $LX'R$ необходимо при вызове **sdmineq** индекс независимой переменной **Xindex** домножить на -1 . При этом нужно помнить, что функция **sdmineq** автоматически добавит в симметрично расположенный блок слагаемое $(LX'R)^*$.

Пример. Для иллюстрации этого замечания рассмотрим два разных вызова функции **sdmineq** [56]:

```
quiz = sdmineq(quiz,[LMIndex blrow blcol], +Xindex, L, R);
quiz = sdmineq(quiz,[LMIndex blcol blrow], -Xindex, R',L');
```

Здесь **quiz** — имя оптимизационной задачи, **LMIndex** — номер неравенства, **Xindex** — номер матричной переменной X . Очевидно в случае вещественных X, L, R эти команды равнозначны, однако для комплексных матриц слагаемые, добавленные этими командами, будут связаны друг с другом комплексным сопряжением.

5. В пакете SeDuMi Interface, как уже было отмечено ранее, есть возможность определения ограничений, в которые входят члены с прямым произведением (произведением Кронекера). С одной стороны, этот оператор упрощает математическую запись неравенства, но с другой стороны, весьма трудоемок с точки зрения программной реализации. Рассматриваемый пакет позволяет избежать эти трудности и непосредственно использовать оператор прямого произведения.

При определении слагаемого, содержащего произведение Кронекера, используются необязательные шестой и седьмой входные параметры **K** и **flag** функции **sdmineq**:

```
pbname = sdmineq(pbname,[lmiindex blrow blcol],
                  Xindex,L,R,K,flag);
```

Если параметр **flag** опущен, в неравенство в симметричные от диагонали блоки добавляются члены вида

$$L(K \otimes X)R, \quad (L(K \otimes X)R)^*.$$

Если же **flag** = -1 , то в неравенство включаются члены вида

$$L(X \otimes K)R, \quad (L(X \otimes K)R)^*.$$

то же время, в языке MATLAB один и тот же символ (штрих) обозначает транспонирование для вещественных матриц и эрмитово сопряжение для комплексных матриц. Для того, чтобы не запутаться, следует учесть, что в математических выражениях мы используем курсив, а в элементах программы — прямой шрифт.

Приведем один из таких примеров [56].

Пример. Рассмотрим неравенства Ляпунова для оценки соответствен-но устойчивости по Гурвицу и по Шуру некоторой матрицы A :

$$A'P + PA < 0, \quad A'PA - P < 0,$$

P — матричная переменная. Оба эти неравенства могут быть представ-лены в общем виде как

$$[IA']K \otimes P \begin{bmatrix} I \\ A \end{bmatrix} < 0,$$

где K соответственно задается по формулам $K = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ и $K = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$. Тогда задание обоих этих неравенств будет выполняться одинаковой ко-мандой

```
>> quiz=sdmineq(quiz,LMIindex,Pindex,[eye(n),A'],[],K).
```

К сожалению, разработчики не предусмотрели возможность провер-ки созданного с помощью функции `sdmineq` неравенства. Функция `get`, которая уже упоминалась выше, может вывести информацию о неравен-ствах только по четырем пунктам: общее число неравенств, имя нера-венства по его индексу, размеры матрицы неравенства и ее минимальное собственное число. В отличие от пакета LMILab, в котором с помощью функции `lmiinfo` можно вывести на экран общий вид заданного нера-венства, SeDuMi Interface такой функции не содержит.

ФУНКЦИЯ `sdmlme`

Задание ограничения в виде равенства, так же как и неравенства, начинается с описания его структуры. Для этого вызывается функция пакета SeDuMi Interface `sdmlme` со следующими аргументами:

```
[pbname,lmeindex]=sdmlme(pbname,dimrow,dimcol,name)
```

Данная функция по своему описанию во многом схожа с функцией `sdmlmi`. В результате вызова `sdmlme` в оптимизационной задаче `pbname` объявляется равенство размеров `dimrow×dimcol` со строкой-описанием `name`, и ему присваивается номер `lmeindex`. Как и для функции `sdmlmi` параметр `name` является необязательным и служит лишь для более удоб-ного вывода информации о равенстве на экран.

Поскольку равенства-ограничения оптимизационной задачи не обязательно являются квадратными, то их размеры определяются двумя параметрами: числом строк `dimrow` и числом столбцов `dimcol`. Значения этих аргументов аналогичны значению параметра `dim` функции `sdmlmi`. А именно, если равенство не имеет блочной структуры, то `dimrow` и `dimcol` — натуральные числа. Если же равенство имеет определенную блочную структуру, то `dimrow` — это массив чисел, равных количеству строк в каждом из диагональных блоков, а `dimcol` — массив чисел, равных количеству столбцов диагональных блоков.

Пример. Рассмотрим ограничение вида

$$\begin{bmatrix} Y & 0 \\ 0 & A \end{bmatrix} v = p,$$

где Y и A — матричные переменные размера $m \times n$ и $m \times m$ соответственно, v и p — заданные векторы размера $m + n$ и $2m$. Очевидно, если $v = [v_1 \ v_2]'$, то имеем

$$\begin{bmatrix} Yv_1 \\ Av_2 \end{bmatrix} = p.$$

Тогда вызов `sdmlme` для инициализации этого равенства будет иметь вид:

```
[LMCpb,Eq1_index]
=sdmlme(LMCpb,[m m],1,'[Y&0 \\\ 0&A] v = p');
```

Теперь поле данных задачи `LMCpb`, отвечающее ограничениям типа равенств, приобретет значение:

```
equality constraints:
      index      norm      name
        1          -Inf      [Y&0 \\\ 0&A] v = p
```

Столбец под названием `norm` содержит значение нормы $\| \cdot \|_1$ (наибольшая по столбцам сумма модулей элементов) ограничения-равенства для имеющихся в системе значений матричных переменных. При инициализации равенства `norm` получает значение $-\infty$.

Функция `sdmeq`

Функция `sdmeq` предназначена для определения конкретного вида ограничения-равенства. Заголовок этой функции следующий:

```
pbname = sdmeq(pbname,lmedata,Xindex,L,R,K,flag)
```

Эта функция является аналогом функции `sdimineq` для неравенств. Приведем краткую характеристику ее параметров, поскольку подробное описание аналогичных параметров было дано выше.

- Параметр `lmeta`. Если он принимает скалярное целое значение (обозначим его `lmeindex`), то слагаемое добавляется к равенству номер `lmeindex`, которое не имеет блочной структуры. Если он представляет собой массив целых чисел `[lmeindex blrow blcol]`, то слагаемое добавляется к равенству номер `lmeindex`, в блок с индексами `(blrow, blcol)`.

Если значение `lmeindex` указывается без знака или со знаком «плюс», то слагаемое добавляется в левую часть равенства, а если со знаком «минус», то в правую.

- Параметр `Xindex` определяет индекс матричной переменной (возможно со знаком или умноженный на число i). Если слагаемое не содержит матричной переменной, то его полагают равным нулю. Пусть, например, индекс `Hindex` используется для описания матрицы H :

если формальному параметру `Xindex` сопоставить фактический параметр `Hindex` или `+Hindex`, то добавляется слагаемое вида LHR ;

если фактический параметр равен `-Hindex`, то добавляется слагаемое вида $LH'R$;

если фактический параметр равен `+i * Hindex`, то добавляется слагаемое вида $L\bar{H}R$;

если фактический параметр равен `-i * Hindex`, то добавляется слагаемое вида LH^*R .

(Еще раз напомним, что штрих обозначает транспонирование, а звездочка — эрмитово сопряжение.)

- Параметры `L`, `R`. Если оба параметра не пустые и не пропущены, то слагаемые добавляются по правилам, приведенным выше. Если параметр `L` не пуст, а параметр `R` — пустой или отсутствует, то по умолчанию он принимает значение $R = L^*$ (эрмитово сопряжение матрицы L). Если оба параметра `L` и `R` пустые или отсутствуют, то они принимают значения единичных матриц, размеры которых согласованы с размерами матричной переменной.

- Параметры K , flag . По умолчанию эти параметры пусты. Пусть X — матрица, отвечающая параметру Xindex . Если параметр K не пуст, а параметр flag пуст, то добавляется слагаемое вида $L(K \otimes X)R$. Если параметр K не пуст, а $\text{flag} = -1$, то добавляется слагаемое вида $L(X \otimes K)R$.

Необходимо подчеркнуть одно существенное различие между ограничениями равенствами и неравенствами. Заключается оно в том, что матрицы, определяющие равенства, не обязательно эрмитовы. Поэтому, добавляя слагаемые к некоторому блоку равенства, нельзя ничего сказать о симметрично-расположенном блоке. Следовательно, все оговорки относительно эрмитово сопряженного слагаемого, добавляемого в симметричный блок, становятся бессмысленными.

Пример. Рассмотрим ограничение-равенство, структура которого определена в предыдущем примере:

$$\begin{bmatrix} Yv_1 \\ A v_2 \end{bmatrix} = p.$$

Для задания ограничения можно выполнить следующую последовательность команд:

```
LMCpb=sdmeq(LMCpb,[Eq1_index 1 1],Y_index,1,v(1:n));
LMCpb=sdmeq(LMCpb,[Eq1_index 2 1],A_index,1,v(n+1:n+m));
LMCpb=sdmeq(LMCpb,-Eq1_index,0,1,p);
```

4.2.4. Функция `sdmobj`

Завершающим шагом в определении оптимизационной задачи является задание линейной целевой функции (linear objective function).

Этой цели служит функция с именем `sdmobj`. Причем в отличие от описанных ранее пакетов, ориентированных на минимизацию заданной целевой функции, SeDuMi Interface предполагает, что ищется решение, доставляющее максимум функции цели. По умолчанию ставится задача исследования разрешимости системы матричных ограничений, а функция цели полагается нулевой.

В случае комплексной целевой функции, решатель максимизирует ее вещественную часть.

Вызов функции `sdmobj` в общем случае имеет вид:

```
pbname=sdmobj(pbname,Xindex,L,R,name)
```

Данная команда в задаче `pbname` добавляет линейный член вида LXR , где L и R — заданные вектор-строка и вектор-столбец соответственно, а X — матричная переменная под номером `Xindex`. Параметр `name`, как и раньше, необязателен и является текстовой строкой-описанием целевой функции. Также необязательным является и четвертый параметр R . Если $R = []$ или он пропущен, то считается, что $R = L'$ (т.е. $R = L^*$).

Интересной особенностью пакета SeDuMi является то, что в качестве целевой функции может использоваться оператор следа матрицы. В такой ситуации левый множитель L полагают равным '`tr`', а R может быть вектором, матрицей или скаляром. Тогда функция `sdmobj` добавляет к целевой функции член $\text{tr}(RX)$. При этом если R пропущен, то по умолчанию $R = 1$.

Пример. Приведем пример вызова функции `sdmobj` для определения оптимизационной задачи с целевой функцией вида [56]

$$\text{tr}(CHC') \rightarrow \max,$$

где C — заданная матрица, H — матричная переменная задачи.

Поскольку справедливо равенство

$$\text{tr}(CHC') = \text{tr}(C'CH),$$

то команда по заданию такой функции цели будет следующей:

```
LMCpb=sdmobj(LMCpb,H_index,'tr',C'*C,'trace CHC');
```

Тогда поле, отражающее информацию о целевой функции примет вид:

```
maximise objective:      trace CHC
```

После задания функции цели, оптимизационная задача полностью определена, и теперь можно переходить к ее непосредственному решению, то есть к вызову программы-решателя.

4.3. Решение оптимизационной задачи

Этот раздел посвящен тому, как пакет SeDuMi Interface позволяет использовать алгоритмы поиска решения оптимизационной задачи, реализованные решателем SeDuMi (речь пойдет о версии SeDuMi 1.05). Кроме того, мы рассмотрим, каким образом интерпретируется полученный результат и настраиваются параметры программы-решателя.

4.3.1. Функция `sdmsol`

Пакет SeDuMi Interface, как следует из его названия, является чисто интерфейсным пакетом и не содержит собственно алгоритмов поиска решения. С помощью функции `sdmsol` он осуществляет связь между удобным способом описания задачи оптимизации, о котором шла речь в этой главе ранее, с внутренним представлением задачи в решателе SeDuMi.

Вызов функции `sdmsol` выглядит следующим образом:

```
[pbname, sdmdata] = sdmsol(pbname, pars, Radius)
```

Здесь `pbname` — имя задачи (объекта класса `sdmpb`), `pars` — переменная типа структуры, чьи поля являются именами настраиваемых параметров решателя, `Radius` — радиус допустимости (feasibility radius) для скалярных переменных. Этот параметр не позволяет вектору скалярных переменных принимать слишком большие (по норме) значения. Последние два параметра являются необязательными.

Если явно не определено устройство вывода текущей информации, функция `sdmsol` отображает состояние задачи на экране. Рассмотрим один из возможных вариантов такого сообщения, при этом нас будет интересовать не конкретное содержание задачи оптимизации, а лишь текст, отображенный на мониторе, и его смысл. Пусть, в результате решения некоторой оптимизационной задачи, появилось текстовое сообщение [56]:

```
>> quiz = sdmsol(quiz);  
  
SeDuMi 1.05 by Jos F. Sturm, 1998, 2001.  
Alg = 2: xz-corrector, Step-Differentiation, theta = 0.250,  
beta = 0.500, eqs m = 16,order n = 13, dim = 69, blocks = 4  
nnz(A) = 74 + 0, nnz(ADA) =256, nnz(L) = 136  
it : b*y gap delta rate t/tP* t/tD* feas cg cg  
0 : 1.73E+07 0.000  
1 : -7.50E-01 1.02E+06 0.000 0.0588 0.0000 0.9000 1.14 1 1  
2 : -1.81E+00 2.66E+05 0.000 0.2614 0.9000 0.9000 0.71 1 1  
3 : -2.91E+00 9.37E+04 0.000 0.3520 0.9000 0.9000 0.28 1 1  
4 : -8.32E+00 2.08E+04 0.000 0.2225 0.9000 0.9121 -0.21 1 1  
5 : -4.47E+01 5.37E+02 0.000 0.0257 0.9000 0.9173 -0.41 1 1  
6 : -2.99E+01 4.72E+01 0.054 0.0879 0.9900 0.9900 1.41 1 1  
7 : -2.82E+01 4.05E+00 0.000 0.0859 0.9000 0.9174 1.00 1 1  
8 : -2.78E+01 1.34E+00 0.000 0.3316 0.9000 0.9000 0.78 1 1  
9 : -2.75E+01 1.43E-01 0.000 0.1067 0.9000 0.9194 0.78 1 1  
10 : -2.74E+01 2.66E-02 0.121 0.1852 0.0000 0.9000 0.70 1 1
```

```

11 : -2.74E+01 7.74E-06 0.000 0.0003 0.9000 0.8498 0.67 1 1
12 : -2.73E+01 1.37E-06 0.000 0.1777 0.8558 0.9000 0.24 2 2
13 : -2.73E+01 9.81E-08 0.000 0.0714 0.9900 0.9900 0.84 2 2
14 : -2.73E+01 3.03E-09 0.003 0.0308 0.9900 0.9900 0.98 2 2
15 : -2.73E+01 1.89E-10 0.000 0.0625 0.9900 0.9900 1.00 2 2
16 : -2.73E+01 3.82E-11 0.000 0.2019 0.9000 0.9000 1.00 3 2
17 : -2.73E+01 1.71E-12 0.000 0.0447 0.9900 0.9900 1.00 3 3

iter seconds digits      c*x          b*y
17      1.1      Inf -2.7287769476e+01 -2.7287769465e+01
|Ax-b| = 1.1e-09, [Ay-c]_+ = 0.0E+00,
|x|= 4.1e+02, |y|= 3.0e+02
Max-norms: ||b||=1, ||c|| = 1,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 193074.
feasible

```

Как видно, сообщение включает в себя большое количество информации, причем по большей своей части она состоит из значений внутренних параметров решателя SeDuMi. Однако нашей целью является обзор пакетов интерфейсного характера, поэтому мы обсудим только некоторые, наиболее полезные из параметров.

Первые несколько строк сообщения содержат входные параметры решателя, которые определены по умолчанию или заданы пользователем самостоятельно (см. раздел 4.3.2). Далее следует блок, выводящий текущие данные о задаче по итерациям. В частности, во втором столбце этого блока **b*y** выводится значение целевой функции двойственной задачи. В столбце под заголовком **gap** приведены значения разрыва двойственности, т.е. разность между целевыми функциями прямой и двойственной задачи. Оптимальные значения обеих целевых функций совпадают, поэтому вычисления прекращаются, когда эта величина становится достаточно малой. Столбец **feas** содержит значение показателя разрешимости (feasibility indicator) задачи. Этот показатель сходится к 1 для разрешимой системы ограничений, к -1 в противном случае. В конце сообщения выводится блок данных о состоянии задачи на оптимальном решении, если таковое было найдено. В том числе этот блок содержит максимальное значение целевой функции двойственной задачи **b*y**, минимальное значение целевой функции **c*x** для прямой задачи оптимизации, а также нормы ограничений прямой и двойственной задач. Последней записью сообщения является вывод о разрешимости поставленной задачи **feasible**. С точки зрения вопроса разрешимости решатель SeDuMi может сделать один из трех выводов:

- **feasible** — означает, что задача разрешима, и при максимизации целевой функции найдена оптимальная точка. В примере оптимум целевой функции равен -27.3 и задача имеет по меньшей мере одно решение, которое и было найдено в процессе работы функции `sdmsol`.
- **infeasible** — означает, что задача неразрешима.
- **marginal feasible** — данный результат появляется в случае, когда в процессе решения возникли какие-либо вычислительные трудности: алгоритм SeDuMi не может сделать вывод о неразрешимости поставленной задачи, но и не может найти оптимальное решение. В этом случае можно попытаться с помощью регулировки параметров решателя настроить алгоритм решения (см. 4.3.2).

Итак, предположим, что поставленная задача решена, и получено оптимальное решение. Естественно, возникает желание проверить, удовлетворяет ли оно заданным неравенствам. Для этой цели можно воспользоваться функцией `sdmpb/get`, а именно вызвать ее с параметром `info = 'ineqmeig'` (раздел 4.5), при этом неравенство считается выполненным, если минимальное собственное значение полученной матрицы положительно. Тогда в результате может быть получен один из следующих ответов:

- `-Inf`, если задача оптимизации еще не решена;
- `0`, если минимальное собственное значение равно нулю (такая ситуация возможна, поскольку в SeDuMi неравенства рассматриваются как полуопределенные);
- `eps` или `-eps`, если минимальное собственное значение (как положительное, так и отрицательное) близко к нулю, по умолчанию точность `eps = 10-9`;
- положительная константа, если неравенство строго выполнено;
- отрицательная константа, если неравенство не выполнено.

4.3.2. Регулировка параметров решателя

Изменение параметров решателя происходит с помощью явного задания значений соответствующих полей структуры `pars`, являющейся одним из аргументов функции `sdmsol`. Полный перечень полей этой структуры с их описанием можно найти в инструкции пользователя пакета

SeDuMi [62], которая к нему прилагается. Мы же остановимся на рассмотрении наиболее полезных параметров.

- Поле **alg**. Позволяет явно указать решателю тип задачи для выбора соответствующего алгоритма решения: 1 — задача полуопределенного программирования, 2 — задача линейного программирования. По умолчанию **alg** = 1.
- Поле **eps**. Задает значение вычислительной точности алгоритма. По умолчанию **eps** = 10^{-9} .
- Поле **fid**. Задает режим вывода информации о вычислениях на экран: 0 — текущая информация на экран не выводится, 1 — на каждой итерации происходит вывод информации о состоянии переменных на экран. В общем случае **fid** — это идентификатор файла или другого устройства. В случае вывода в файл, идентификатору **fid** присваивается значение, полученное при открытии файла с помощью функции **fopen**.
- Поле **maxiter**. Устанавливает максимально возможное число итераций. По умолчанию **maxiter** = 50.
- Поле **Radius**. Определяет верхнюю границу для нормы вектора скалярных переменных задачи. Явное задание этого параметра добавляет еще одно ограничение к оптимизационной задаче. Если $0 < \text{Radius} < \text{Inf}$, то добавляется ограничение сверху для нормы вектора скалярных неизвестных. Если **Radius** = 0, к системе ограничений добавляется фиктивное ограничение. Если **Radius** = **Inf**, никаких ограничений не добавляется. По умолчанию **Radius** = 0.

Пример. В качестве иллюстрации влияния выбора значения параметра **pars** на полученное решение рассмотрим пример, приведенный авторами пакета [56]. Решается смешанная H_2/H_∞ задача синтеза обратной связи по состоянию, порядок системы $n = 10$. Предположим, что заданы все компоненты задачи **quiz** и вызывается решатель SeDuMi со значением **pars** по умолчанию. Вот какое сообщение будет получено в итоге:

```
>> qiu=sdmsol(quiz);

LMC problem: H2/Hinfty state-feedback synthesis
matrix variable:           index    name
                      1          X
```

```
          2      T
          3      S
no equality constraint
inequality constraints: index  meig    name
                         1      0.005  X>0
                         2      3e-09   Hinfty
                         3      3e-08   gram
                         4      -eps    H2
maximise objective:-trace(T) -g = -3.86
feasible
```

Алгоритм SeDuMi дает однозначный ответ: задача разрешима, то есть найденное решение — оптимальное. Тем не менее, матрица четвертого неравенства на этом оптимальном решении имеет отрицательное минимальное собственное значение. Такая ситуация означает, что алгоритм остановился, достигнув заданной по умолчанию точности `eps` вычисления оптимума целевой функции, однако этой точности не хватило, чтобы удовлетворить одно из неравенств. Выход — изменение значения `eps`:

```
>> pars.eps=1e-12;
>> qiu=sdmsol(quiz);

LMC problem: H2/Hinfty state-feedback synthesis
matrix variable:           index   name
                     1       X
                     2       T
                     3       S
no equality constraint
inequality constraints: index  meig    name
                           1      0.005  X>0
                           2      3e-09   Hinfty
                           3      3e-08   gram
                           4      -7e-10  H2
maximise objective:-trace(T) -g = -3.86
marginal feasible
```

Теперь, несмотря на то, что минимальное собственное значение матрицы последнего неравенства все равно осталось отрицательным, алгоритм SeDuMi сделал более адекватный вывод: «*marginal feasible*». Тем не менее, такой вывод неудовлетворителен, поскольку свидетельствует скорее о вычислительных проблемах алгоритма, чем о существе поставленной задачи.

Авторы пакета предлагают два варианта действий в такой ситуации:

1. Первый путь вытекает из того факта, что в SeDuMi все неравенства считаются полуопределенными. Тогда для того, чтобы при решении было удовлетворено строгое неравенство, необходимо, чтобы минимальное собственное значение `meig` матрицы этого неравенства было положительной константой. Если же `meig` равно нулю или близко к нему, предлагается модифицировать неравенства, добавив постоянный член $-\alpha I_p$ (I_p — единичная матрица соответствующего размера, $\alpha > 0$) в правую часть неравенства:

$$L(x) \leq R(x) - \alpha I_p \quad \rightarrow \quad L(x) < R(x).$$

В рассматриваемом примере при $\alpha = 1e-06$ такое изменение неравенств может быть получено командой

```
>> quiz2=quiz-1e-06
```

Решая модифицированную задачу `quiz2`, получим:

```
>> qiu2=sdmsol(quiz2);

LMC problem: H2/Hinf state-feedback synthesis
matrix variable:      index    name
                  1          X
                  2          T
                  3          S
no equality constraint
inequality constraints: index    meig    name
                         1       0.005   X>0
                         2       eps     Hinfty
                         3       7e-09  gram
                         4      -eps    H2
maximise objective:-trace(T) -g = -3.86
feasible
```

Таким образом, получен конкретный ответ относительно разрешимости поставленной задачи, и собственные значения заданных неравенств превосходят величину $\alpha - \text{eps} = 10^{-6} - 10^{-9} > 0$.

2. Другой путь решения задачи `quiz` — это явное задание третьего аргумента функции `sdmsol` (радиуса разрешимости `Radius`):

```
>> qiuZ=sdmsol(quiz,[ ],1e9);

LMC problem: H2/Hinfty state-feedback synthesis
matrix variable:      index   name
                  1        X
                  2        T
                  3        S
no equality constraint
inequality constraints: index   meig    name
                         1       0.01    X>0
                         2       6e-06   Hinfty
                         3       2e-05   gram
                         4       8e-07   H2
maximise objective:-trace(T) -g = -3.86
feasible
```

Отметим, что в качестве нижней границы значения параметра *Radius* выступает норма вектора решения, полученного без явного задания радиуса разрешимости.

4.4. Модификация задачи: функция *sdmset*

Возможность модифицировать существующую задачу является показателем гибкости интерфейсного пакета. SeDuMi Interface в определенном смысле обладает таким свойством: он содержит функцию *sdmset*, которая позволяет присвоить фиксированное значение некоторой матричной переменной. Эту функцию, в частности, используют при проверке принадлежности определенного значения матричной переменной области допустимых значений.

Вызов функции выглядит так:

```
pbname = sdmset(pbname, Xindex, Xvalue)
```

Здесь *pbname* — имя модифицируемой задачи, *Xindex* — индекс той матричной переменной, которой присваивается значение *Xvalue*.

Необходимо отметить, что действие данной функции не распространяется на матричные переменные, которые определены через другие матричные переменные.

Пример. Объявим задачу и матричные переменные *H*, *A*, найдем их структурные матрицы:

```

LMCpb = sdmpb('test');
[LMCpb,H_index]=sdmvar(LMCpb,3*i,'h','H');
[LMCpb,A_index]=sdmvar(LMCpb,3,'as','A');
H_struct=get(LMCpb,'vardec',H_index);
A_struct=get(LMCpb,'vardec',A_index);

```

Сформируем матричную переменную U из имеющихся блоков по формуле

$$U = [\bar{H} \ A]$$

и выведем структурные матрицы переменных H и U на экран:

```

U_nbrow=[conj(H_struct), A_struct];
[LMCpb,U_index]=sdmvar(LMCpb,U_nbrow,'st','U');
U_struct=get(LMCpb,'vardec',U_index);
H_str=num2str(H_struct)
U_str=num2str(U_struct)

```

В результате получим:

```

H_str =
1+0i   2-2i   3-3i
2+2i   4+0i   5-5i
3+3i   5+5i   6+0i

U_str =
1+0i   2+2i   3+3i   0+0i   -7+0i   -8+0i
2-2i   4+0i   5+5i   7+0i   0+0i   -9+0i
3-3i   5-5i   6+0i   8+0i   9+0i   0+0i

```

Теперь присвоим значение переменной H и снова выведем структурные матрицы:

```

H0=[1 0 0; 0 1 0; 0 0 1];
LMCpb=sdmset(LMCpb,H_index,H0);
H_struct=get(LMCpb,'vardec',H_index);
U_struct=get(LMCpb,'vardec',U_index);
H_new=num2str(H_struct)
U_new=num2str(U_struct)

```

Получаем:

```

warning : index refers to a variable that was set
to a constant

```

```
H_new =
,,

U_new =
1+0i   2+2i   3+3i   0+0i   -7+0i   -8+0i
2-2i   4+0i   5+5i   7+0i   0+0i   -9+0i
3-3i   5-5i   6+0i   8+0i   9+0i   0+0i
```

Таким образом, мы получили предупреждение о том, что матричной переменной *H* присвоено постоянное значение. Ее структурная матрица пуста, так как *H* больше не зависит от скалярных переменных. Структура переменной *U* не изменилась после присваивания матрице *H* значения *H0*.

4.5. Получение информации о задаче: функция `get`

Для получения информации о созданной оптимизационной задаче, хранящейся в виде объекта класса `smpb`, служит функция `smpb/get`. Вызов этой функции выглядит следующим образом:

```
out = get(pbname,info,index)
```

Здесь `pbname` — имя задачи, `info` — строка, определяющая какая именно информация будет выведена на экран, `index` — индекс той компоненты задачи, о которой необходимо получить информацию (индекс матричной переменной, матричного неравенства или равенства). Значение выходного параметра `out` зависит от параметра `info` и приведено в табл. 4.3.

Таблица 4.3.

| Значение <code>info</code> | Значение <code>out</code> |
|-------------------------------|--|
| [] | <code>out = []</code> — на экран выводится текст встроенной помощи функции <code>get</code> . |
| 'name' | Имя объекта класса <code>smpb</code> , в котором содержится информация о задаче оптимизации. |
| 'varnb' | Число объявленных матричных переменных задачи. |
| 'vardecnb' | Число скалярных переменных задачи. |

Продолжение табл. 4.3.

| Значение <code>info</code> | Значение <code>out</code> |
|-------------------------------|---|
| <code>'varname'</code> | Имя матричной переменной с индексом <code>index</code> . |
| <code>'varvalue'</code> | Значение матричной переменной X с индексом <code>index</code> . (Может быть получено только после вызова функции <code>sdmsol</code> , в противном случае выдается сообщение об ошибке.) |
| <code>'vardec'</code> | Структурная матрица переменной X с индексом <code>index</code> . Размер матрицы <code>out</code> совпадает с размером переменной X , а элементы определены следующим образом: $\text{out}(k, j) = 0$, если $X(k, j) = 0$; $\text{out}(k, j) = +n$, если элемент $X(k, j)$ вещественный и равен n -ой скалярной переменной; $\text{out}(k, j) = -n$, если $X(k, j)$ вещественный и равен $(-1)*n$ -ю скалярную переменную; $\text{out}(k, j) = +n + n * i$, если $X(k, j)$ комплексный и равен n -ой скалярной переменной; $\text{out}(k, j) = -n - n * i$, если $X(k, j)$ комплексный и равен $(-1)*n$ -ю скалярную переменную; $\text{out}(k, j) = +n - n * i$, если $X(k, j)$ комплексный и равен комплексно сопряженной n -ой скалярной переменной; $\text{out}(k, j) = -n + n * i$, если $X(k, j)$ комплексный и равен $(-1)*$ комплексно-сопряженную n -ю скалярную переменную. |
| <code>'ineqnb'</code> | Число ограничений-неравенств задачи. |
| <code>'ineqname'</code> | Имя ограничения-неравенства с индексом <code>index</code> . |
| <code>'ineqdim'</code> | Вектор, состоящий из размеров диагональных блоков в неравенстве с индексом <code>index</code> . |
| <code>'ineqmeig'</code> | Минимальное собственное значение матрицы неравенства с индексом <code>index</code> . Если задача была решена, то ограничение считается выполненным при $\text{meig} \geq 0$. |

Окончание табл. 4.3.

| Значение <code>info</code> | Значение <code>out</code> |
|-------------------------------|--|
| 'eqnb' | Число ограничений-равенств задачи. |
| 'eqname' | Имя ограничения-равенства с индексом <code>index</code> . |
| 'eqdimrow' | Вектор размерностей по строкам блоков матрицы ограничения-равенства с индексом <code>index</code> . |
| 'eqdimcol' | Вектор размерностей по столбцам блоков матрицы ограничения-равенства с индексом <code>index</code> . |
| 'eqnorm' | Норма ограничения-равенства с индексом <code>index</code> . Если задача была решена с помощью функции <code>sdmsol</code> , то ограничение считается выполненным при <code>mnorm ≤ eps</code> . |
| 'objname' | Имя целевой функции задачи. |
| 'objopt' | Оптимальное значение целевой функции (при условии, что был осуществлен вызов функции <code>sdmsol</code>). |
| 'solver' | Содержит информацию о сходимости процедуры максимизации целевой функции (при условии, что был осуществлен вызов функции <code>sdmsol</code>). |
| 'ynorm' | Норма вектора решения (при условии, что был осуществлен вызов функции <code>sdmsol</code>). |
| 'feas' | <code>out = 1</code> , если задача разрешима и найдено оптимальное решение; <code>out = 0</code> , если задача условно разрешима (<i>marginal feasible</i>), то есть алгоритм SeDuMi не может сделать вывод о неразрешимости задачи, но при этом возникли вычислительные трудности при нахождении оптимального решения. В этом случае алгоритм решения модифицируется с помощью изменения параметров решателя (см. раздел 4.3.2); <code>out = -1</code> , если матричные ограничения несовместимы. |

Некоторое неудобство пакета SeDuMi Interface состоит в отсутствии способа проверки правильности сформированных ограничений. Для сравнения вспомним функцию `lmiinfo` пакета LMILab (глава 2), которая выводит информацию как обо всей матрице неравенства, так и об отдельном ее блоке. К сожалению, пакет SeDuMi Interface такой возможности не имеет. Этот недостаток весьма заметен, поскольку затрудняет процесс анализа полученного решения.

Примеры применения функции `get` были уже даны в разделах 4.2.2 и 4.3.1.

4.6. Примеры использования пакета

Как мы уже отмечали, пакет SeDuMi Interface является достаточно полным аналогом пакета LMILab (а во многом и расширяет его возможности). Поэтому для иллюстрации применения этого пакета рассмотрим примеры из главы 2.

4.6.1. Круговой критерий абсолютной устойчивости

Проверка кругового критерия абсолютной устойчивости нелинейной системы управления (2.3), (2.4) сводится к проверке разрешимости системы линейных матричных неравенств (2.7) относительно симметричной матрицы H .

Приведем текст функции `sdm_circle`, чье действие полностью совпадает с функцией `circle` (см. раздел 2.7.1), но которая использует при этом средства пакета SeDuMi Interface. Напомним, что входными параметрами являются матрица A и вектора b , c , описывающие систему управления в пространстве состояний, а μ_1 , μ_2 — заданные вещественные числа из соотношения (2.4). На выходе `sdm_circle` выдает показатель разрешимости задачи `flag`, а также одно из значений матрицы H , удовлетворяющее ограничениям задачи.

```
function [flag,H0] = sdm_circle(A,b,c,mu1,mu2)
%Circle criterion
sz=size(A);
CC=sdmpb('Circle criterion');
[CC,H]=sdmvar(CC,sz(1),'s','H');
[CC,MLMI]
    =sdmlmi(CC,[sz(1) 1],'Main LMI');
CC=sdmineq(CC,[MLMI 1 1],H,1,A);
CC=sdmineq(CC,[MLMI 1 2],H,1,b);
```

```
CC=sdmineq(CC, [-MLMI 1 1], 0, 1, 0.5*mu1*mu2*c*c');
CC=sdmineq(CC, [-MLMI 1 2], 0, 1, -0.5*(mu1+mu2)*c);
CC=sdmineq(CC, [-MLMI 2 2], 0, 1);
[CC,Hpos]=sdmlmi(CC,sz(1), 'H>0');
CC=sdmineq(CC, -Hpos, H);
CC=sdmsol(CC);
H1=get(CC, 'varvalue', H);
H0=double(H1);
flag=get(CC, 'feas');
```

Как видно, различия двух сравниваемых пакетов минимальны, за исключением одного факта. Если при добавлении некоторого члена к диагональному блоку неравенства пакет SeDuMi Interface автоматически добавляет транспонированное комплексно-сопряженное слагаемое, то LMI-lab этого не делает.

Используем функцию `ctest`, описанную в разделе 2.7.1, для тестирования функции `sdm_circle`. Тогда результат действия последней на том же тестовом примере будет следующий:

```
SeDuMi 1.05 by Jos F. Sturm, 1998, 2001. Alg = 2: xz-corrector,
Step-Differentiation, theta = 0.250, beta = 0.500
eqs m = 4, order n = 8, dim = 18, blocks = 4
nnz(A) = 15 + 0, nnz(ADA) = 16, nnz(L)= 10
it :    b*y      gap    delta   rate   t/tP*   t/tD*   feas   cg   cg
0 :          5.01E+007 0.000
1 : 0.00E+000 3.77E+006 0.000 0.0754 0.9900 0.9901 1.40   1   1
2 : 0.00E+000 1.34E+005 0.000 0.0354 0.9000 0.9105 1.34   1   1
3 : 0.00E+000 1.29E-003 0.189 0.0000 0.9990 0.9986 1.10   1   1
4 : 0.00E+000 8.84E-006 0.072 0.0069 0.9990 0.7588 1.23   1   1
5 : 0.00E+000 1.27E-008 0.000 0.0014 0.9990 0.9990 1.00   1   1
6 : 0.00E+000 8.84E-015 0.000 0.0000 1.0000 1.0000 1.00   1   1
iter seconds digits      c*x            b*y
6       0.1   5.0  1.6257802330e-015  0.00000000000e+000
|Ax-b| = 1.8e-015, |Ay-c|_+ = 0.0E+000,
|x|= 6.7e-015, |y|= 2.5e+000
Max-norms: ||b||=0, ||c|| = 1, Cholesky |add|=0,
|skip| = 0, ||L.L|| = 1.5567. feasible
```

Problem is feasible

H0 =

$$\begin{array}{cc} 0.9935 & 0.2912 \\ 0.2912 & 0.1810 \end{array}$$

4.6.2. Оценка коэффициента диссипативности

Обсуждение проблемы диссипативности в общем виде можно найти, например, в [61]. Следуя [35], здесь мы рассмотрим одну из частных постановок этой задачи. Пусть задана линейная система управления

$$\frac{dx}{dt} = Ax + Bu, \quad y = C'x + Du, \quad (4.1)$$

где x — n -мерный вектор состояний, u и y — векторы входа и выхода соответственно, их размерности совпадают и равны m . Говорят, что система (4.1) имеет диссипацию $\eta > 0$, если при нулевых начальных условиях $x(0) = 0$ для всех допустимых входов $u(t)$ и любого числа $T > 0$ вход и выход системы удовлетворяют интегральному неравенству:

$$\int_0^T (u(t)'y(t) - \eta u(t)'u(t)) dt \geq 0. \quad (4.2)$$

Наибольшая диссипация, т.е решение задачи $\eta \rightarrow \max$, называется коэффициентом диссипативности системы (4.1) [35]. Если система (4.1) управляема и наблюдаема, то

$$\eta_{\max} = \frac{1}{2} \inf_{\text{Re } p > 0} \lambda_{\min}(W(p) + W(p)^*),$$

где $W(p)$ — передаточная матрица системы.

Задача нахождения коэффициента диссипативности может быть сведена к задаче полуопределенного программирования:

$$\begin{aligned} \eta &\rightarrow \max, \\ \begin{bmatrix} HA + A'H & HB - C \\ B'H - C' & 2\eta I_m - D' - D \end{bmatrix} &\leq 0, \quad H > 0, \quad \eta > 0. \end{aligned} \quad (4.3)$$

Действительно, если определить функцию $V(x) = x'Hx$, то из ограничений задачи (4.3) следует, что производная $V(x)$ в силу системы (4.1) удовлетворяет неравенству

$$\frac{dV}{dt} - 2u'y + 2\eta u'u \leq 0,$$

откуда следует (4.2).

Опишем функцию для определения коэффициента диссипативности:

```
function [kdiss, H1, flag] = dissip(A,B,C,D)
% Dissipation
sz=size(B);
n=sz(1);
m=sz(2);
kdiss = [ ];
H1 = [ ];
Cpb = sdmpb('dissipation');
[Cpb,H_index] = sdmvar(Cpb, n, 's', 'H');
[Cpb,eta_index] = sdmvar(Cpb, 1, 1, 'eta');

[Cpb,lmi1_index]=sdmlmi(Cpb,[n m], 'Main inequality');
Cpb=sdmineq(Cpb, [lmi1_index 1 1],H_index,1,A);
Cpb=sdmineq(Cpb, [lmi1_index 1 2],H_index,1,B);
Cpb=sdmineq(Cpb, [-lmi1_index 1 2], 0, 1, C);
Cpb=sdmineq(Cpb, [lmi1_index 2 2], eta_index,1,eye(m));
Cpb=sdmineq(Cpb, [-lmi1_index 2 2], 0, 1, D);

[Cpb,lmi2_index] = sdmlmi(Cpb,n,'H>0');
Cpb=sdmineq(Cpb, -lmi2_index, H_index);

[Cpb,lmi3_index] = sdmlmi(Cpb,1,'eta>0');
Cpb=sdmineq(Cpb, -lmi3_index, eta_index);

Cpb=sdmobj(Cpb,eta_index,1,1,'eta max');

pars.fid = 0;
Cpb = sdmsol(Cpb,pars);

flag = get(Cpb,'feas');
if flag == 1 || flag == 0
    H0 = get(Cpb,'varvalue',H_index);
    H1 = double(H0);
    k0 = get(Cpb,'objopt');
    kdiss = double(k0);
end
```

Выходными параметрами функции являются `kdiss` — коэффициент диссипативности, `H1` — матрица H , `flag` — характеризует успешность решения задачи. Полагая поле `pars.fid` равным нулю, мы блокируем вывод служебной информации решателя SeDuMi. В конце работы функции вы-

полнянем преобразование типа данных `sdmvar` в `double`. Вызовем функцию с помощью сценария:

```
clc;
numerator = [1 2];
denominator = [1 3];
w = tf(numerator,denominator);
sys = ss(w);
[A,B,C,D] = ssdata(sys);
C = C';
[kdiss,H,flag]=dissip(A,B,C,D);
if flag == 1
    disp('Problem is feasible');
    H, kdiss
elseif flag == 0
    disp('Problem is marginal feasible');
    H, kdiss
else
    disp('Problem is not feasible');
end
```

Здесь $W(p) = (p + 2)/(p + 3)$. Получаем:

```
Problem is feasible
H =
    1.0000

kdiss =
    1.6667
```

Непосредственным вычислением легко проверить, что точное значение `kdiss` равно $5/3$.

Глава 5. Пакет YALMIP

5.1. Общая характеристика пакета

Пакет YALMIP (Yet Another LMI Parser) разработан Юханом Леффергом из университета Линчепинга, Швеция (в настоящее время работает в Федеральном техническом институте, Цюрих, Швейцария). Он предназначен для системы MATLAB, может работать в операционных системах Windows и Solaris. В январе 2004 г. была выпущена версия 3 пакета — на нее мы и будем ориентироваться. Пакет бесплатный и может быть загружен со страницы <http://control.ee.ethz.ch/joloef/yalmip.mysql>. В связи с тем, что сейчас в YALMIP постоянно вносятся исправления и обновления, его пользователям рекомендуется следить за новостями на сайте Ю. Лефферга.

Особенность этого интерфейсного пакета — большое число программ-решателей, которые он поддерживает. К ним относятся SeDuMi, SDPT3, DSDP, CSDP, SDPA, MAXDET, GLPK. В версии 3 добавлен интерфейс с пакетами LINPROG, QUADPROG, PENSDP, PENBMI, LMILab, SDPLR, CDD, NAG, CPLEX, KYPD.¹ При этом прекращена поддержка пакетов SP, SOCP, MoSeK и старых версий пакетов SDPT3, CSDP.

Некоторые имена функций различных решателей конфликтуют друг с другом. Например, не следует устанавливать одновременно SDPT3 и SeDuMi.

В руководстве пользователя Ю. Лефферг дает краткие характеристики некоторым программам-решателям. В первую очередь он рекомендует SeDuMi как достаточно быстрый, обладающий хорошей устойчивостью и точностью результатов. Решатель DSDP имеет смысл использовать в случае сильной разреженности матричных коэффициентов. При работе с CSDP применяется запись данных задачи во вспомогательный файл, что несколько замедляет работу в случае большого количества данных. Для работы с этим пакетом следует использовать его версию SDPA-M.

¹Работа с KYPD подробно описана в следующей главе.

Для применения MAXDET необходимо загрузить его модифицированную версию с сайта пакета YALMIP. Хотя использование LMILab и допускается, Ю. Лефберг решительно не рекомендует использовать его совместно с YALMIP.

Сам автор признает, что YALMIP несколько проигрывает в скорости выполнения другим известным пакетам, если число скалярных переменных превышает несколько сотен. Кроме того, YALMIP иногда дает сбои при работе с плохо обусловленными задачами, особенно большой размерности. Круг задач, решаемых с помощью YALMIP, достаточно широк. В этой главе мы не будем рассматривать возможности YALMIP по исследованию задач линейного и квадратичного программирования, так как наши интересы ограничиваются вопросами полуопределенного программирования. Кроме того, опущено описание ряда вспомогательных и второстепенных функций.

Как и Sedumi Interface, пакет YALMIP имеет объектную организацию. Он включает классы `sdpvar`, `lmi`, `constraint`, `logdet`, а также ряд внешних функций.

Поясним несколько запутанную ситуацию с функцией `set`. В какой-то момент разработчику не понравилось имя функции (и класса) `lmi`, и в версии 3 он решил заменить его на `set`.² Чтобы не вносить никаких изменений в существующий программный код, он просто описал функцию `set`, которая вызывает конструктор класса `lmi`. Таким образом, `set` не является классом, хотя внешне и выглядит таковым. В документации к пакету упоминается то «класс `set`», то «класс `lmi`» (по-видимому, там, где разработчик еще не успел внести исправления). Сознавая, что большинству пользователей нет необходимости вдаваться в подобные тонкости, мы, следуя разработчику, будем говорить о классе `set`.

5.2. Описание матричных переменных

Как и в остальных интерфейсных пакетах, создание системы линейных матричных неравенств распадается на два этапа: первый — определение матричных переменных, второй — задание ограничений. В пакете YALMIP матричные переменные являются объектами класса `sdpvar` (если требуется скалярная переменная, то она определяется как матрица размера 1×1).

²Эту идею нельзя признать удачной, так как в языке MATLAB уже имеется стандартная функция `set`, с помощью которой можно изменять значения полей объекта. Никакого отношения к «классу `set`» эта функция не имеет.

Матричная переменная создается вызовом конструктора класса `sdpvar`. Имя конструктора совпадает с именем класса, т.е. функция-конструктор также имеет имя `sdpvar`. Заголовок конструктора имеет вид

```
X = sdpvar(m,n,matrixtype,datatype)
```

Здесь `m` — число строк и `n` — число столбцов. Последние три параметра не обязательны. Если указан только один параметр, то матрица считается квадратной. Параметр `matrixtype` характеризует тип матрицы и может принимать значения:

- '`symmetric`' — симметрична,
- '`full`' — заполненная (произвольная),
- '`hankel`' — ганкелева,
- '`toeplitz`' — тэплицева,
- '`skew`' — кососимметрична (антисимметрична),
- '`hermitian`' — эрмитова.

В случае типа '`full`' элементы матрицы не связаны никакими соотношениями. По умолчанию задается значение '`symmetric`', если $n = m$, и '`full`' в противном случае. Можно использовать сокращенные значения: '`sy`', '`f`', '`ha`', '`t`', '`sk`', '`he`'. Параметр `datatype` определяет тип элементов матрицы и принимает только два значения:

- '`real`' — вещественная,
- '`complex`' — комплексная

(по умолчанию принимается '`real`').

Пример. Определим симметричную 5×5 -матрицу H :

```
H = sdpvar(5);
```

Определим комплексную матрицу K размером 2×3 произвольной структуры:

```
K = sdpvar(2, 3, 'full', 'complex');
```

Кроме конструктора класс `sdpvar` имеет более тридцати методов. Их полный список можно получить с помощью команды

```
methods('sdpvar')
```

Большинство из них задает перегрузку (переопределение) стандартных функций и операций языка MATLAB, предназначенных для работы с матрицами. В частности перегружены операции сложения, вычитания, умножения матриц, операции отношения, операции транспонирования и индексации, функции `trace`, `sum`, `diag` и др. Таким образом, работа с

матричными переменными, которые являются объектами класса `sdpvar`, почти не отличается от работы со стандартными матрицами.³

Для создания матричных переменных, все элементы которых целые или принимают значения 0 и 1, вместо функции `sdpvar` следует использовать функции `intvar` или `binvar`.

Для присваивания значений матричным переменным может быть использована функция `setsdpvar`. Ее вызов имеет вид:

```
setsdpvar(X,value)
```

Здесь `X` — объект класса `sdpvar`, `value` — значение типа `double` (т.е. обычная матрица). Отметим, что, если скалярные переменные, входящие в `X`, входят и в другие матричные переменные, то эти матричные переменные также будут изменены. Обычно `setsdpvar` используют перед вызовом решателя для задания начальных приближений.

Пример. Последовательность команд

```
P = sdpvar(3);
setsdpvar(P,eye(3));
double(P)
```

создает симметричную вещественную матрицу порядка три и инициализирует ее единичной матрицей. Последняя команда вызывает операцию преобразования типа `sdpvar` в `double`. В результате на экран будет выведена единичная матрица.

Функция `fixsdpvar` позволяет зафиксировать часть скалярных переменных, входящих в матричную переменную, задав их постоянные значения. Вызов функции

```
Z = fixsdpvar(X,Y,W)
```

заменяет вхождения объекта `Y` (класса `sdpvar`) в объект `X` на постоянное значение `W`.

Пример. В результате последовательности команд

```
P = sdpvar(3);
P = fixsdpvar(P,diag(P),[1;1;1]);
```

диагональные элементы переменной `P` будут зафиксированы и равны единице.

³В языке MATLAB стандартные матрицы имеют тип `double`.

Пример. Если определить

```
P = sdpvar(3);  
P = fixsdpvar(P,P(1,3),0);
```

то элемент переменной P с индексами $(1, 3)$ и симметричный ему элемент с индексами $(3, 1)$ будут зафиксированы и равны нулю.

Пример. В документации к пакету приведен следующий пример работы с «составными» переменными:

```
x = sdpvar(1); y = sdpvar(1); z = sdpvar(1);  
s = [x y z];  
s = fixsdpvar(s,[y z],[1 2]);
```

5.3. Создание системы ограничений

В задачах, которые решает YALMIP, могут использоваться два вида ограничений: ограничения типа неравенств и ограничения типа равенств. При этом YALMIP не делает различий между строгими и нестрогими неравенствами — все неравенства понимаются как нестрогие. Тем не менее, при решении задачи полуопределенного программирования имеется возможность обеспечить строгость ограничений типа неравенств. Для этого служит специальный параметр `'shift'` (см. следующий раздел). Все ограничения хранятся в объекте класса `set`, причем каждое ограничение получает порядковый номер, начиная от единицы.

При вызове функции `set` без параметров

```
F = set([ ]);
```

создается «пустая» задача, ограничения в которую могут быть добавлены позднее.

При описании ограничений можно использовать знаки операций отношений «больше», «меньше», «равно» ($>$, $<$, $==$). В случае сравнения симметричных и эрмитовых матриц операции $>$ и $<$ понимаются в смысле положительной (отрицательной) определенности, в остальных случаях — как поэлементное сравнение.

Пример. Создадим переменную H типа симметричной матрицы размера 3×3 :

```
H = sdpvar(3);
```

Ограничение $H > 0$ (которое означает, что матрица H положительно определена) задается следующим образом:

```
F = set(H>0);
```

Отметим, что неравенство $H \geq 0$ задается точно так же, т.е. YALMIP не делает различия между строгими и нестрогими неравенствами, причем операции \geq и \leq (так же как и логические операции) при задании ограничений не допускаются.

Для того, чтобы потребовать положительность всех элементов матрицы H , следует задать ограничение:

```
F = set(H(:)>0);
```

Теперь определим прямоугольную (не квадратную) матрицу

```
M = sdpvar(5,6);
```

Тогда ограничение

```
F = set(M>0);
```

будет означать уже не положительную определенность, а положительность всех элементов матрицы M .

Пример. Ограничение $HA + A'H < 0$, где A — квадратная матрица, можно задать в виде:

```
F = set(H * A + A'* H < 0);
```

Пример. Ограничение $Hb = c$, где b, c — матрицы, задается в виде

```
F = set(H * b == c);
```

Напомним некоторые особенности операций отношения в языке MATLAB. (Теми же свойствами обладают и перегруженные операции «больше», «меньше», «равно» для объектов класса `sdpvar`). Сравнивать можно либо матрицы одного размера, либо матрицу и скалярное значение. В остальных случаях при несовпадении размеров матриц-операндов, выдается сообщение об ошибке. Если один из operandов скалярный, а другой — матричный, то перед сравнением скаляр распространяется до матрицы подходящего размера, все элементы которой одинаковы. Таким образом, можно задать ограничения

```
H=sdpvar(2);
F=set(H<1);
```

Оно означает, что

$$H < \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Предыдущие примеры показывают, как задавать одиночные ограничения (в виде одного неравенства или равенства). Покажем на примерах как могут быть составлены более сложные ограничения.

Пример. Пусть требуется задать систему ограничений вида

$$H > 0, \quad HA + A'H < 0, \quad Hb = c.$$

1-й способ:

```
F=set(H * A + A'* H < 0) + set(H * b == c) + set(H > 0);
```

2-й способ:

```
F = set(H * A + A'* H < 0);
F = F + set(H * b == c);
F = F + set(H > 0);
```

В этих двух случаях используется операция сложения, перегруженная для объектов класса `set`. Кроме сложения можно использовать также функции `mergelmi`, `addlmi`, которые объявлены устаревшими и сохраняются лишь для совместимости версий пакета.

Кроме того, для задания нескольких ограничений можно использовать двойные неравенства, например,

```
F = set(0 < H < eye(3));
```

означает, что $0 < H < I_3$, где I_3 — единичная матрица порядка 3.

Можно потребовать, чтобы элементы матричной переменной принимали только целые значения или значения 0 и 1:

```
F = set(integer(H));
F = set(binary(H));
```

Наряду с использованием перегруженных операций «больше», «меньше», «равно» можно задавать ограничения в виде строк. Подробное описание такого способа задания можно найти в руководстве по пакету YAL-MIP. Например:

```
F = set('H * A + A''* H < 0');
```

(внутри строки знак транспонирования следует удвоить).

5.4. Решение задачи полуопределенного программирования

В пакете YALMIP имеется возможность решения нескольких различных задач оптимизации. Мы опишем только одну из этих задач — задачу полуопределенного программирования. Для этой цели используется функция `solvesdp`. Это внешняя функция, т.е. она не является методом какого-либо класса. Ее вызов имеет вид

```
diagnostics = solvesdp(F,h,options)
```

Здесь все три входные параметра необязательны.⁴

`F` — объект класса `set`, содержащий описание ограничений.

`h` — объект класса `sdpvar` или `logdet`, описывающий целевую функцию. В процессе решения эта функция минимизируется. Параметр `h` может быть опущен, если исследуется задача разрешимости системы ограничений (feasibility problem), а не оптимизации.

`options` — структура, описывающая параметры оптимизации. Если положить

```
options = sdpsettings;
```

то будут использованы параметры, задаваемые по умолчанию. Для изменения стандартных параметров используется конструкция вида

```
options = sdpsettings('name1',value1,'name2',value2,...);
```

т.е. указывается имя параметра (`name`) в виде строки и его значение (`value`). Для всех параметров, не заданных явно, сохраняются их стандартные значения. Для того, чтобы внести изменения в существующую структуру `oldoptions`, функцию `sdpsettings` можно вызвать в виде

```
newoptions = sdpsettings(oldoptions,'name1',value1,...);
```

Укажем наиболее важные из параметров.

`'solver'` — имя программы-решателя. Для решения задач полуопределенного программирования могут быть использованы значения

```
'sdpt3', 'sedumi', 'sdpa', 'pensdp', 'penbmi',
'csdp', 'dsdp', 'maxdet', 'lmilab', 'sdplr', 'kypd'.
```

⁴Если необязательный параметр является последним, он может быть опущен. Необязательный параметр, стоящий в начале или в середине списка параметров, может быть заменен пустыми квадратными скобками [].

(Остальные значения отвечают решателям, используемым для других классов задач.) Если решатель не указан явно, то функция `solvesdp` сама пытается подобрать подходящий решатель из числа, установленных в системе MATLAB.

`'verbose'` — характеризует уровень подробности сообщений выдаваемых решателем. Может принимать целые неотрицательные значения (0 — минимальное число сообщений, 1 — стандартные сообщения, 2 и более — подробные сообщения). Значение по умолчанию равно 1.

`'radius'` — вещественное число, которое задает ограничения для векторов, составленных из скалярных переменных (вида $\|x\| < R$). По умолчанию равен `Inf` (т.е. бесконечности в системе MATLAB).

`'saveduals'` — может принимать значения 0 или 1 (1 по умолчанию). Единица означает, что при решении следует сохранить значения вспомогательных двойственных переменных. Эти значения могут быть извлечены из описания задачи с помощью функции `dual`.

`'shift'` — неотрицательное вещественное число (по умолчанию 0). Используется в случае, когда необходимо обеспечить выполнение строгих ограничений типа неравенств. Этот параметр характеризует возмущение исходной системы неравенств с тем, чтобы «отстроиться» от нуля. Естественно, его следует выбирать достаточно малым.

`'removeequalities'` — с помощью этого режима можно на этапе предварительной обработки (до вызова решателя) исключить ограничения типа равенств. Это особенно существенно для решателей, которые не допускают такой вид ограничений (например, `lmilab`). Параметр может принимать значения -1, 0, 1 (по умолчанию 0). Значение -1 предписывает заменить равенство на два неравенства, 0 — не выполнять никаких преобразований, 1 — выполнить QR-разложение.

`'usex0'` — число 0 или 1 (0 по умолчанию). Единица означает, что данные до вызова решателя значения переменных следует использовать в качестве начальных приближений. Для задания начальных значений переменных можно применить функцию `setsdpvar`.

Выходной параметр `diagnostics` представляет собой структуру, содержащую диагностические сообщения. Она имеет следующие поля:

`yalmiptime` — время (в секундах), затраченное функциями пакета YALMIP на решение задачи (без учета времени работы внешнего решателя);

`solvertime` — время работы решателя;

`info` — формируемый функцией `yalmiperror` текст сообщения об ошибке (если ошибки нет, обычно выводится сообщение `No problems detected`);

`problem` — код ошибки (целое число от -5 до 13); при нормальной работе имеет нулевое значение (расшифровку кода можно найти в описании функции `yalmiperror`).

Оптимальные значения целевой функции и матричных переменных могут быть извлечены из этих же переменных с помощью функции `double`:

```
sys = double(P)
```

Здесь `P` — объект класса `sdpvar`.

Возникает естественный вопрос — каким образом результаты работы функции `solvesdp` хранятся в переменных, которые не являются выходными параметрами этой функции? Дело в том, что конструктор класса `sdpvar` содержит локальные статические переменные, т.е. переменные, значения которых сохраняются между вызовами функции.⁵ Различные функции пакета вызывают конструктор `sdpvar` не для создания новых объектов, а для чтения или изменения этих переменных. Именно в статических переменных и хранятся результаты работы функции `solvesdp`.

Для получения значений двойственных переменных используется упомянутая ранее функция `dual`:

```
sys = dual(F(n))
```

(разумеется, она применима только для таких решателей, которые решают двойственную задачу). Здесь `F` — объект класса `set`, т.е. массив ограничений, `F(n)` — ограничение номер `n`. (Напомним, что ограничения нумеруются, начиная с единицы.) С каждым таким ограничением связывается двойственная матричная переменная. Функция `dual` служит для получения значений этих переменных.

Для того, чтобы количественно оценить насколько хорошо полученное решение удовлетворяет заданным ограничениям, служит функция `checkset`. Функция может вызываться либо в виде

```
checkset(F)
```

т.е. без указания выходных параметров, либо в виде

```
[P,D] = checkset(F)
```

В первом случае результаты ее работы выводятся на экран в виде таблицы. Во втором — выходной параметр `P` характеризует ограничения

⁵В языке MATLAB локальные статические переменные описываются с помощью оператора `persistent`.

прямой задачи, а выходной параметр D — ограничения двойственной задачи. В качестве параметра функции F может использоваться либо объект класса **set**, т.е. массив ограничений, либо выражение вида $F(n)$, где n — номер ограничения.

Выдаваемая функцией информация зависит от типа ограничений. Для ограничения прямой задачи вида $F(x) > 0$, где неравенство понимается в смысле квадратичных или эрмитовых форм, функция **checkset** возвращает минимальное собственное значение $F(x)$. Если неравенство $F(x) > 0$ понимается как поэлементное, она возвращает минимальный элемент матрицы $F(x)$. Для ограничения типа равенства $F(x) = 0$ функция возвращает максимальный из модулей элементов $F(x)$. Аналогичная информация выдается для двойственных ограничений.

5.5. Примеры исследования линейных матричных неравенств

5.5.1. Круговой критерий абсолютной устойчивости

Рассмотрим функцию, предназначенную для проверки кругового критерия абсолютной устойчивости (см. раздел 2.7.1). Ее параметры имеют тот же смысл, что и в примере, рассмотренном в 2.7.1.

```
function [flag,H1] = circle(A,b,c,mu1,mu2)
% Circle criterion
sz=size(A);
n = sz(1);
flag=0;
H1=[];
H=sdpvar(n);
R=[mu1*mu2*c*c' -0.5*(mu1+mu2)*c; -0.5*(mu1+mu2)*c' 1];
F=set([H*A+A'*H H*b; b'*H 0]< R)+ set(H>0);
options = sdpsettings('solver','sedumi','verbose',0);
diagn=solvesdp(F,[],options)
if diagn.problem == 0
    flag=1;
    H1=double(H);
end
```

Из этого примера видно, что YALMIP требует явного задания всех элементов неравенства, тогда как для других пакетов (LMILab, SeDuMi Interface) достаточно задать элементы над или под главной диагональю.

В данном примере ограничения достаточно простые и это обстоятельство не слишком существенно, однако для сложных неравенств объем «лишнего» программного кода может оказаться весьма значительным.

5.5.2. Оценка H_∞ -нормы системы управления

Рассмотрим задачу об оценке коэффициента усиления по норме пространства L_2 (L_2 -gain) [41]. Пусть линейная стационарная система описывается уравнениями

$$\frac{dx}{dt} = Ax + bu, \quad y = c'x, \quad (5.1)$$

где x — вектор состояния системы, u и y — вектора входа и выхода, A , b , c — матричные коэффициенты подходящих размеров. Предположим, что вектор-функция $u(t)$ суммируема с квадратом на интервале $[0, +\infty)$, т.е. $u \in L_2[0, +\infty)$. Тогда она имеет конечную L_2 -норму

$$\|u\|_2 = \left[\int_0^\infty u'(t)u(t) dt \right]^{1/2}.$$

Если матрица A гурвицева, то из $u \in L_2[0, +\infty)$ следует $y \in L_2[0, +\infty)$ при любых начальных данных. Нас интересует величина L_2 -усиления:

$$\sup_{\|u\|_2 \neq 0} \frac{\|y\|_2}{\|u\|_2},$$

где супремум берется по $u \in L_2[0, +\infty)$ и предполагается, что $x(0) = 0$.

Для устойчивых систем усиление по L_2 -норме совпадает с H_∞ -нормой линейной системы, которая в скалярном случае определяется как $\sup_\omega |W(i\omega)|$, а в векторном случае — как $\sup_\omega \sigma_{\max}(W(i\omega))$. Здесь $W(p)$ — передаточная функция (в векторном случае — матрица) системы, $\sigma_{\max}(M)$ — наибольшее сингулярное число матрицы M . Доказательство этого факта можно найти, например, в [8]. Заметим, что функции для вычисления H_∞ -нормы имеются в нескольких стандартных пакетах MATLAB (например, функция `norm`, входящая в Control System Toolbox, или `norminf` из LMI Control Toolbox). Насколько можно судить по документации, в основном они используют не методы внутренней точки, а исследование вспомогательной гамильтоновой матрицы [30].

Допустим, что симметричная матрица H и положительное число γ удовлетворяют линейному матричному неравенству

$$\begin{bmatrix} HA + A'H & Hb \\ b'H & 0 \end{bmatrix} + \begin{bmatrix} cc' & 0 \\ 0 & -\gamma^2 I \end{bmatrix} \leq 0.$$

Рассмотрим квадратичную форму $V(x) = x' H x$. Рассматривая ее производную в силу системы (5.1), из матричного неравенства получим, что

$$\frac{dV}{dt} + y'y - \gamma^2 u'u \leq 0.$$

Проинтегрируем это неравенство от нуля до $+\infty$. Если $x(0) = 0$ и $x(t) \rightarrow 0$ при $t \rightarrow +\infty$, то $\gamma \|u\|_2 \geq \|y\|_2$, т.е. γ дает оценку сверху L_2 -усиления. Полагая $t = \gamma^2$, приходим к следующей оптимизационной задаче вида (6.1):

$$\begin{aligned} t &\rightarrow \min, \\ \begin{bmatrix} HA + A'H & Hb \\ b'H & 0 \end{bmatrix} + \begin{bmatrix} cc' & 0 \\ 0 & -tI \end{bmatrix} &\leq 0. \end{aligned}$$

Заметим, что если выполнено либо неравенство $HA + A'H < 0$, либо неравенство $HA + A'H \leq -cc'$, где пара матриц (A, c) наблюдаема, то гурвицевость матрицы A эквивалентна неравенству $H > 0$ [6].

Приведем функцию, решающую задачу нахождения L_2 -усиления:

```
function [r,H] = l2gain1(A,b,c)
% L2-gain
szb = size(b);
n = szb(1);
m = szb(2);
t = sdpvar(1);
P = sdpvar(n);
F = set([P*A+A'*P+c*c' P*b; b'*P -t*eye(m)]<0);
F = F+set(P>0);
options = sdpsettings('solver','sedumi');
solvesdp(F,t,options);
r = sqrt(double(t));
H = double(P);
```

5.6. Использование ограничений типа билинейных матричных неравенств

В ноябре 2003 г. в рамках проекта PENOPT был выпущены пакеты программ-решателей PENSDP и PENBMI [47, 48] (разработчики Michal Kočvara и Michael Stingl, Universität Erlangen–Nürnberg, Германия). Пакеты имеют интерфейс с языками программирования C/C++, Fortran и

MATLAB. Версия 3 YALMIP обеспечивает для этих пакетов возможность работы с матричными переменными и удобное задание матричных коэффициентов. Комплексы программ PENOPT и TOMLAB, которые включают PENSDP и PENBMI, являются коммерческими, поэтому, к сожалению, они недоступны большинству российских пользователей. Тем не менее мы разберем работу (в рамках YALMIP) с пакетом PENBMI, поскольку он рассматривает несколько отличный от других пакетов класс задач и, по-видимому, в настоящее время не имеет аналогов.

В PENBMI решается задача полуопределенного программирования с квадратичной целевой функцией, в качестве ограничений выступают билинейные матричные неравенства (bilinear matrix inequalities, BMI). В несколько упрощенном виде задача заключается в следующем:

$$\begin{aligned} & \frac{1}{2}x'Qx + c'x \rightarrow \min, \\ & F_0 + \sum_{i=1}^m x_i F_i + \sum_{j=1}^m \sum_{k=1}^m x_j x_k G_{jk} \leq 0. \end{aligned}$$

Последнее неравенство понимается в смысле отрицательной полуопределенности матрицы. Здесь x — вектор скалярных переменных, Q , F_i , G_{jk} — симметричные матрицы, c — вектор. Описание реализованного в PENBMI алгоритма (penalty/barrier multiplier method — PBM) и обзор существующих подходов к исследованию BMI можно найти в [42]. (Обсуждение различных аспектов работы с BMI содержится также в [29], глава 14.) Заметим, что имеющиеся алгоритмы решения BMI еще далеки от совершенства. В частности, алгоритм PBM позволяет решать задачу полуопределенного программирования только локально. Существующий метод глобального исследования (вариант метода ветвей и границ) в виде программного обеспечения пока еще не реализован. Теоретические исследования показывают, что глобальные алгоритмы решения оптимизационных задач с BMI должны иметь сложность выше полиномиальной (класса NP), поэтому от методов решения билинейных матричных неравенств не следует ожидать столь высокой эффективности как от методов, ориентированных на линейные матричные неравенства. Тем не менее, не исключено, что со временем эффективные алгоритмы глобальной оптимизации будут развиты для каких-то специальных классов BMI.

Билинейные ограничения встречаются в теории управления достаточно часто. Примером может служить система неравенств

$$\begin{bmatrix} H(A + \lambda I_n) + (A' + \lambda I_n)H & Hb \\ b'H & 0 \end{bmatrix} \leq F, \quad H + \tau c c' > 0, \quad \lambda > 0, \quad \tau > 0,$$

где H — неизвестная симметричная матрица, λ и τ — неизвестные вещественные числа, A, b, c, F — матричные коэффициенты подходящих размеров [6]. Левая часть первого неравенства билинейна (но не линейна) по совокупности коэффициентов H, λ . Формально эту задачу можно рассматривать как обобщенную проблему собственных чисел (GEVP), но, например, пакет LMILab здесь неприменим, так как матрица коэффициентов при неизвестной λ вырождена.

Заметим, что анонсирован выпуск в ближайшем будущем еще одного пакета, входящего в комплекс PENOPT, под названием PENNON. Он также может быть использован для исследования билинейных матричных неравенств.

Виду того, что в нашем распоряжении нет коммерческого пакета PEN-BMI, мы ограничимся слегка измененным демонстрационным примером, который входит в дистрибутив YALMIP.

Пример. Рассмотрим систему управления

$$\frac{dx}{dt} = Ax + Bu + Gw, \quad y = Cx, \quad (5.2)$$

где u, w и y — скалярные функции, x — n -вектор. Обобщим рассмотренную ранее задачу о вычислении L_2 -усиления следующим образом. Пусть y — выход системы, w — ее внешний вход, а u — ее внутренний вход (в рассматриваемом демонстрационном примере они предполагаются скалярными, что несущественно). Требуется подобрать линейную стационарную обратную связь вида $u = Lx$, где L — неизвестный вектор-строка, таким образом, чтобы замкнутая система была устойчивой и L_2 -усиление от w к y было наименьшим. Проблема сводится к оптимизационной задаче:

$$\begin{aligned} t &\rightarrow \min, \\ \begin{bmatrix} H(A + BL) + (A + BL)'H + C'C & HG \\ G'H & -t \end{bmatrix} &< 0, \quad H > 0. \end{aligned} \quad (5.3)$$

Здесь H — неизвестная симметричная матрица, L — неизвестный вектор-строка, t — неизвестный скаляр.

Приведем демонстрационную программу, решающую эту задачу:

```
% Синтезируем систему по передаточной функции
P = ss(tf(1,[1 1 1 1]));
A = P.a;
B = P.b;
G = P.b;
C = P.c;
```

```
% Определяем переменные
H = sdpvar(3);
L = sdpvar(1,3);
t = sdpvar(1);

% Присваиваем переменным начальные значения
[L0,H0]=lqr(A,B,eye(3),1);
setsdpvar(H,H0);
setsdpvar(L,-L0);
setsdpvar(t,100);

% Определяем BMI
F = set([(A+B*L)'*P+P*(A+B*L)+C'*C P*G;G'*P -t]<0) + set(P>0);

% Решаем задачу, используя заданные начальные значения P, L, t
solvesdp(F,t,sdpsettings('solver','penbmi','usex0',1))
double(L)
```

Поясним процедуру выбора начальных приближений для матричных переменных. Стандартная функция MATLAB `lqr` служит для решения линейно-квадратичной задачи оптимального управления на бесконечном интервале времени. В данном случае для линейной системы

$$\frac{dx}{dt} = Ax + Bu$$

решается задача нахождения стабилизирующей обратной связи вида $u = -Kx$, которая минимизирует функционал

$$J(x, u) = \int_0^{+\infty} (x'x + u^2) dt.$$

Вектор коэффициентов усиления K находится по формуле $K = B'P$, где P — симметричное решение квадратичного матричного уравнения

$$I_n + PA + A'P - PBB'P = 0$$

такое, что матрица $A - BB'P$ гурвицева. В качестве начальных значений H и L берутся оптимальные значения P и $-K$.

Заметим, что исходная задача может быть решена и с помощью более традиционных линейных матричных неравенств, но при этом потребуются дополнительные усилия по преобразованию системы [35]. Используя

формулы Шура (см. главу 1), сведем основное неравенство задачи (5.3) к квадратичному матричному неравенству

$$H(A + BL) + (A + BL)'H + C'C + t^{-1}HGG'H < 0.$$

Обозначив $P = tH^{-1}$, $Y = LP$, получим

$$PA' + AP + BY + Y'B' + GG' + PC'CP + GG' < 0.$$

Вновь применяя формулы Шура, перейдем к эквивалентному неравенству

$$\begin{bmatrix} PA' + AP + BY + Y'B' + GG' & PC' \\ CP & -t \end{bmatrix} < 0,$$

линейному относительно неизвестных P , Y и t .

Часть пакета YALMIP, посвященная ВМП, в настоящее время активно развивается, поэтому читателю, заинтересованному в этом классе задач, рекомендуется следить за сайтом YALMIP.

5.7. Вспомогательные команды

Перечислим некоторые полезные команды пакета, носящие вспомогательный характер.

`yalmipdemo` — выводит на экран набор демонстрационных примеров.

`yalmiptest` — тестирует MATLAB на наличие установленных решателей. После этого проверяет их работоспособность.

`yalmip('clear')` — очищает рабочее пространство, удаляя из него объекты классов `set` и `sdpvar`.

`yalmip('nvars')` — возвращает число скалярных переменных, присутствующих в рабочем пространстве.

`yalmip('info')` — выводит информацию о переменных и ограничениях, присутствующих в рабочем пространстве.

`yalmip('solver', s)` — устанавливает в качестве решателя по умолчанию решатель с именем `s` (`s` — строка, ее возможные значения перечислены при описании функции `sdpsettings`). Например:

```
yalmip('solver', 'sedumi');
```

Глава 6. Пакет KYPD

6.1. Общая характеристика пакета

KYPD — небольшой и совсем новый пакет (2003 г.), разработанный аспирантом университета Линчепинга (Швеция) Рагнаром Валлином (Ragnar Wallin). В конце января 2004 г. была выпущена версия 1.2, доступная по адресу <http://www.control.isy.liu.se/publications/doc?id=1470>. Общая документация к пакету содержится в [66], технические детали можно найти в [41]. Для работы KYPD требуется одновременная установка пакетов SeDuMi и YALMIP, т.е. KYPD является надстройкой над пакетом YALMIP и использует решатель SeDuMi.¹

Пакет KYPD предназначен для решения задач полуопределенного программирования, связанных с леммой Калмана—Якубовича—Попова. В отечественной литературе это утверждение чаще называют частотной теоремой или леммой Якубовича—Калмана [6, 20]. Точнее, с помощью функций пакета решается задача

$$\begin{aligned} c'x &\rightarrow \min, \\ F_i(x, H_i) &\geq 0, \quad i = 1, 2, \dots, N, \end{aligned} \tag{6.1}$$

где

$$F_i(x, H_i) = \begin{bmatrix} H_i A_i + A'_i H_i & H_i B_i \\ B'_i H_i & 0 \end{bmatrix} + M_{i0} + \sum_{j=1}^K x_j M_{ij}.$$

Здесь $c \in \mathbb{R}^K$, $A_i \in \mathbb{R}^{n_i \times n_i}$, $B_i \in \mathbb{R}^{n_i \times m_i}$, матрицы M_{ij} симметричны. Как указано в [41], для задач рассматриваемого класса число скалярных переменных может получиться очень значительным, поэтому непосредственное применение стандартных решателей не всегда возможно. Основная цель создания пакета KYPD — за счет предварительной обработки уменьшить число переменных двойственной задачи, после этого

¹Интересно, что если пакет SeDuMi не установлен, вызывается решатель пакета LMILab, но получаемый при этом результат заведомо неверен.

методы одновременного решения прямой и двойственной задачи могут быть применены более эффективно.

Пакет содержит всего две функции — `kypd` и `kypd_solver`. Функцию `kypd` следует использовать, если заранее известно, что матрицы M_{ij} линейно независимы, пары матриц (A_i, B_i) управляемы, а матрицы A_i — гурвицевы. В остальных случаях нужно применять `kypd_solver`.

Поскольку в качестве решателя используется SeDuMi, при вызове функций пакета одновременно решаются прямая и двойственная задачи. При вызове `kypd_solver` исходная задача подвергается дополнительной обработке: за счет понижения размерности задачи достигаются свойства управляемости и линейной независимости матриц M_{ij} , за счет введения стабилизирующей местной обратной связи обеспечивается гурвицевость коэффициентов A_i .

6.2. Описание основных функций

Две основные функции пакета вызываются следующим образом:

```
[u,H,x,Z] = kypd(matrix_info, options)
[u,H,x,Z] = kypd_solver(matrix_info, options)
```

Все сведения о коэффициентах исходной задачи хранятся в структуре `matrix_info`, которая имеет следующие поля:

`matrix_info.N` — число ограничений;
`matrix_info.K` — размерность вектора x (число скалярных переменных прямой задачи);
`matrix_info.c` — вектор c , состоящий из коэффициентов целевой функции;
`matrix_info.A{i}` — матрица A_i ;
`matrix_info.B{i}` — матрица B_i ;
`matrix_info.M0{i}` — матрица M_{i0} ;
`matrix_info.M{i,j}` — матрица M_{ij} ;
`matrix_info.n` — вектор-столбец, составленный из чисел n_i , т.е.
 $n = [n_1, n_2, \dots, n_N]'$;
`matrix_info.nm` — вектор-столбец, равный

$$nm = [n_1 + m_1, n_2 + m_2, \dots, n_N + m_N]'$$

Отметим, что поля `A`, `B`, `M`, `M0` представляют собой массивы ячеек.

Параметр `options` представляет собой структуру, которая задает режимы решателя и предварительной обработки. Параметр является необязательным; если он опущен, то

```
options = sdpsettings;
```

где `sdpsettings` — функция пакета YALMIP. Таким образом, режимы можно задавать двумя способами — с помощью параметра `options`, например,

```
options.tol = 1e-6;
```

и в стиле пакета YALMIP, например,

```
sdpsettings('kypd.tol', 1e-6);
```

Общими для обеих функций `kypd` и `kypd_solver` являются режимы:

`tol` — параметр, который определяет точность вычисления решения и отвечает параметру `eps` пакета SeDuMi (значение по умолчанию 10^{-8});

`maxiters` — параметр пакета SeDuMi, определяющий максимальное число итераций (значение по умолчанию 100);

`luariponvsolver` — может принимать значения '`diago`' или '`schur`', определяет способ решения вспомогательных уравнений Ляпунова (в первом случае матрицы A_i приводятся к диагональной форме, во втором — к треугольной), значение по умолчанию — '`schur`';

`reduce` — может принимать значения 0 или 1 (по умолчанию — 0), значение 1 означает, что перед вызовом решателя из двойственной задачи устраняются некоторые ограничения типа равенств; при этом размерность задачи понижается, но зато она может лишиться некоторых других свойств, например, разреженности (см. детали в [41]).

При вызове функции `kypd_solver` используются еще два поля структуры `options`:

`transform` — принимает значения 0 или 1, по умолчанию — 0; если параметр равен нулю, то вспомогательная обратная связь вводится только в случае, когда матрица A_i не является гурвицевой; если параметр равен единице, то вспомогательная обратная связь вводится в любом случае, с целью улучшения свойств сходимости.

`rho` — неотрицательный параметр ρ , используемый при введении вспомогательной обратной связи (по умолчанию равен единице).

Остановимся подробнее на смысле последних параметров. Рассмотрим линейную подсистему управления

$$\dot{x}_i = A_i x_i + b_i u_i.$$

Найдем для нее обратную связь по состоянию $u_i = -L_i x_i$, где L_i — постоянная матрица, такая, что замкнутая система

$$\dot{x}_i = (A_i - B_i L_i) x_i$$

устойчива (т.е. матрица $A_i - B_i L_i$ — гурвицева) и на решениях замкнутой системы достигается минимум критерия качества:

$$J = \int_0^\infty (x'_i x_i + \rho u'_i u_i) dt.$$

Хорошо известно (см., например, [1]), что оптимальное значение L_i может быть найдено как $L_i = B'_i S_i / \rho$, где S_i — симметричная матрица, которая является положительно определенным решением алгебраического уравнения Риккати

$$S_i A_i + A'_i S_i - \rho S_i B_i B'_i S_i + I_{n_i} = 0.$$

Заменим неравенство $F_i(x, H_i) \geq 0$ на эквивалентное неравенство

$$\begin{bmatrix} I_{n_i} & 0 \\ -L_i & I_{m_i} \end{bmatrix}' F_i(x, H_i) \begin{bmatrix} I_{n_i} & 0 \\ -L_i & I_{m_i} \end{bmatrix} \geq 0.$$

Последнее неравенство имеет ту же структуру, что и исходное, но матрица A_i в нем заменена на $A_i - B_i L_i$.

Рассмотрим теперь выходные параметры функций:

u — значение целевой функции $c'x$;

H — массив матриц H , полученных на последней итерации;

x — вектор x , полученный на последней итерации;

Z — массив решений Z_i двойственной задачи, полученных на последней итерации.

Пример. Рассмотрим задачу оценки коэффициента усиления по норме пространства L_2 , подробно описанную в предыдущей главе. Напоминаем, что она сводится к оптимизационной задаче

$$\begin{aligned} t &\rightarrow \min, \\ \begin{bmatrix} HA + A'H & Hb \\ b'H & 0 \end{bmatrix} + \begin{bmatrix} -cc' & 0 \\ 0 & tI \end{bmatrix} &\geq 0. \end{aligned}$$

Заметим, что по сравнению с примером предыдущей главы здесь знак неравенства изменен на противоположный. В результате матрица H получается не положительно, а отрицательно определенной. Другое отличие — при использовании КУПД не требуется явно накладывать ограничение $H > 0$ (или $H < 0$).

Приведем функцию, решающую поставленную задачу:

```

function r = l2gain3(A,b,c)
% L2-gain
szb = size(b);
m_info.N = 1;
m_info.K = 1;
m_info.c = 1;
m_info.A{1} = A;
m_info.B{1} = b;
m_info.M0{1} = blkdiag(-c*c', zeros(szb(2)));
m_info.M{1,1} = blkdiag(zeros(size(A)), eye(szb(2)));
m_info.n = [szb(1)];
m_info.nm = [szb(1) + szb(2)];
[u,H,x,Z] = kypd_solver(m_info);
r = sqrt(u);

```

6.3. Работа с KYPD в рамках пакета YALMIP

Мы предполагаем, что до чтения этого параграфа читатель уже познакомился с главой, посвященной пакету YALMIP. Как указывалось ранее, пакет KYPD изначально был ориентирован на работу вместе с YALMIP. В предыдущем параграфе показано как можно вызывать оригинальные функции KYPD. В качестве альтернативы можно использовать функцию `solvesdp` пакета YALMIP, указывая KYPD в качестве решателя. (Хотя, строго говоря, KYPD представляет собой не решатель, а только препроцессор.) В этом случае матричные переменные должны быть определены с помощью стандартной функции YALMIP `sdpvar`, а для задания ограничений можно использовать функцию YALMIP `kyp`. Эта функция вызывается как

```
sys = kyp(A,B,H,M);
```

и служит для создания матрицы

$$\begin{bmatrix} HA + A'H & HB \\ B'H & 0 \end{bmatrix} + M.$$

Пример. Приведем новый вариант текста функции для нахождения L_2 -усиления:

```

function [r,H] = l2gain4(A,b,c)
% L2-gain
szb = size(b);

```

```
n = szb(1);
m = szb(2);
t = sdpvar(1);
P = sdpvar(n);
F = set(kyp(A,b,P,blkdiag(c*c', -t*eye(m))) < 0);
options = sdpsettings('solver','kypd');
solvesdp(F,t,options);
r = sqrt(double(t));
H = double(P);
```

Приложение 1. Установка дополнительных пакетов в системе MATLAB

В системе MATLAB при работе с любыми описанными в этой книге пакетами, кроме стандартного пакета LMILab, требуется сделать дополнительные установки путей. Обычно пути устанавливаются ко всем папкам пакета, кроме папок с документацией. Заметим, что пути к папкам, содержащим описания классов (имена этих папок начинаются с символа @), никогда не указываются непосредственно. Для работы с классом должен быть установлен путь к родительской папке, в которой находится папка описания класса.

MATLAB хранит информацию о путях к установленным пакетам в файле

```
.\toolbox\local\pathdef.m
```

Модификацию этого файла можно выполнить интерактивно или программно (с помощью функций пакета MATLAB).

Проще выполнить установку путей интерактивно. В MATLAB 6.5 для этого нужно проделать следующее. Выберем пункт меню **File\SetPath**. Появляется диалоговое окно, в правой части которого — список установленных путей, а в левой и внизу — ряд командных кнопок. Для добавления нового пути следует нажать либо кнопку **Add Folder** (добавить папку) или **Add with Subfolders** (добавить папку вместе с вложенными в нее папками). Так как оптимизационные пакеты имеют довольно сложную структуру, обычно следует воспользоваться второй кнопкой. Появляется новое диалоговое окно с заголовком «Обзор папок» («Browse for Folder»), содержащее дерево папок. Выберем в нем папку, путь до которой мы хотим установить, и нажмем кнопку **OK**. Пути к папке (и вложенным в нее папкам) будут добавлены в начало списка путей. Если мы хотим поместить их не в начало, а в какое либо другое место списка, можно воспользоваться кнопками **Move Down**, **Move Up** (сдвинуть вниз, сдвинуть вверх) или **Move to Top**, **Move to Bottom** (переместить

в верхнюю или нижнюю часть списка). Если мы хотим сохранить конфигурацию путей для последующих сеансов работы, следует нажать кнопку **Save**. Для удаления путей из списка служит кнопка **Remove**.

Интерактивный способ корректировки списка путей является наиболее простым и наглядным, он может быть рекомендован большинству пользователей. При применении программного способа можно выдать команду **addpath** или добавить вызовы функции **path** в файл **startup.m**. Подробнее об этом можно прочесть в документации по системе MATLAB.

Не рекомендуется устанавливать одновременно много решателей и интерфейсных пакетов, так как имена функций в некоторых из них конфликтуют друг с другом. При этом найти причину конфликта бывает достаточно трудно. Следует учитывать, что при совпадении имен функций вызывается та из них, которая входит в пакет, раньше указанный в файле **pathdef.m**.

Приложение 2. Интернет–адреса основных пакетов, посвященных LMI и SDP

| Пакет | Адрес |
|-------------------|---|
| CSDP | http://www.nmt.edu/~borchers/csdp.html |
| DSDP | http://www-unix.mcs.anl.gov/~benson/ |
| KYPD | http://www.control.isy.liu.se/publications/doc?id=1470 |
| LMILab Translator | http://vehicle.me.berkeley.edu/~guiness/lmitrans.html |
| LMISol | http://www.dt.fee.unicamp.br/~mauricio/lmisol10.html |
| LMITOOL | http://robotics.eecs.berkeley.edu/~elghaoui |
| MAXDET | http://www.stanford.edu/~boyd/MAXDET/ |
| PENSDP, PENBMI | http://www.penopt.com |
| SDPA | http://www.is.titech.ac.jp/~yamashi9/sdpa/index.html |
| SDPHA | http://members-http-3.rwc1.sfba.home.net/nathan-brixius/SDPHA/ |
| SDPpack | http://cs.nyu.edu/cs/faculty/overton/sdppack/sdppack.html |
| SDPSOL | http://www.stanford.edu/~boyd/SDPSOL/ |
| SDPT3 | http://www.math.nus.edu.sg/~mattohkc/ |
| SeDuMi | http://fewcal.kub.nl/sturm/software/sedumi.html |
| SeDuMi Interface | http://www.laas.fr/~peaucell/SeDuMiInt.html |
| SOCP | http://www.stanford.edu/~boyd/SOCP/ |
| YALMIP | http://control.ee.ethz.ch/~joloef/yalmip.msdl |
| xLMI | http://legend.me.uiuc.edu/dullerud/Master/Software/sw_main_xlmtools.html |

Литература

1. *Андреев Ю.Н.* Управление конечномерными линейными объектами. М.: Наука, 1976.
2. *Андреевский Б.Р., Фрадков А.Л.* Избранные главы теории автоматического управления с примерами на языке MATLAB. СПб.: Наука, 1999.
3. *Андреевский Б.Р., Фрадков А.Л.* Элементы математического моделирования в программных средах MATLAB 5 и Scilab. СПб.: Наука, 2001.
4. *Васильев Ф.П.* Численные методы решения экстремальных задач. М.: Наука, 1980.
5. *Гантмахер Ф.Р.* Теория матриц. М.: Наука, 1967.
6. *Гелиг А.Х., Леонов Г.А., Якубович В.А.* Устойчивость нелинейных систем с неединственным состоянием равновесия. М.: Наука, 1978.
7. *Гилл Ф., Мюррей У., Райт М.* Практическая оптимизация. М.: Мир, 1985.
8. *Дезоэр Ч., Видьясагар М.* Системы с обратной связью: вход-выходные соотношения. М.: Наука, 1983.
9. *Дьяконов В.П., Круглов В.В.* Математические пакеты расширения MATLAB. Специальный справочник. СПб.: Питер, 2001.
10. *Зангвишл У.* Нелинейное программирование. М.: Советское радио, 1973.
11. *Лурье А.И.* Некоторые нелинейные задачи теории автоматического регулирования. М.: Гостехиздат, 1951.
12. *Ляпунов А.М.* Общая задача об устойчивости движения. М.: Гостехиздат, 1950.

13. *Магарил-Ильяев Г.Г., Тихомиров В.М.* Выпуклый анализ и его приложения. М.: УРСС, 2003.
14. *Мартынов Н.Н.* Введение в MATLAB 6. М.: КУДИЦ-ОБРАЗ, 2002.
15. *Медведев В.С., Потемкин В.Г.* Control System Toolbox. MATLAB 5 для студентов. М.: Диалог-МИФИ, 1999.
16. *Мусеев Н.Н., Иванилов Ю.П., Столярова Е.М.* Методы оптимизации. М.: Наука, 1978.
17. *Павлова М.И.* Руководство по работе с пакетом SCILAB. 2003. http://www.csa.ru/~zebra/my_scilab/index.html.
18. *Полак Э.* Численные методы оптимизации. М.: Мир, 1974.
19. *Поляк Б.Т., Щербаков П.С.* Робастная устойчивость и управление. М.: Наука, 2002.
20. *Попов В.М.* Гиперустойчивость автоматических систем. М.: Наука, 1970.
21. *Потемкин В.Г.* MATLAB 5 для студентов. М.: Диалог-МИФИ, 1998.
22. *Потемкин В.Г.* MATLAB 6: Среда проектирования инженерных приложений. М.: Диалог-МИФИ, 2003.
23. *Пятницкий Е.С., Скородинский В.И.* Численные методы построения функций Ляпунова и критериев абсолютной устойчивости в форме численных процедур // Автоматика и телемеханика. 1983. № 11. С. 52–63.
24. *Фиакко А., Мак-Кормик Г.П.* Нелинейное программирование. Методы последовательной безусловной минимизации. М.: Мир, 1972.
25. *Чурилов А.Н.* Об оценках функционала, встречающегося при исследовании дискретных систем управления // Известия вузов. Математика. 1984. № 9. С. 59–65.
26. *Якубович В.А.* Решение некоторых матричных неравенств, встречающихся в теории автоматического регулирования // Докл. АН СССР. 1962. Т. 143. № 6. С. 1304–1307.
27. *Якубович В.А.* S-процедура в нелинейной теории регулирования // Вестник ЛГУ, сер. 1. 1971. Вып. 1 (№ 1). С. 62–77.

28. Якубович В.А. Частотная теорема в теории управления // Сибирский математ. журн. 1973. Т. 14. № 2. С. 384–420.
29. Advances in Linear Matrix Inequality Methods in Control / L. El Ghaoui and S.-I. Niculescu (editors). Philadelphia: SIAM, 1999.
30. Balas G.J., Doyle J.C., Glover K., Packard A., Smith R. μ -Analysis and Synthesis Toolbox. User's Guide. Version 3. Natick, MA: The MathWorks, Inc., 1998.
31. Barabanov N.E., Prokhorov D.V. Stability analysis of discrete-time recurrent neural networks // IEEE Trans. Neural Networks. 2002. V. 13. № 2. P. 292–303.
32. Ben-Tal A., Nemirovski A. Lectures on Modern Convex Optimization: Analysis, Algorithms, Engineering Applications. Philadelphia: SIAM, 2001.
33. Ben-Tal A., Zibulevsky M. Penalty/barrier multiplier methods for convex programming problems // SIAM J. Optimization. 1997. V. 7. № 2. P. 347–366.
34. Boyd S., Barrat C. Linear Controller Design: Limits of Performance. Englewood Cliffs, NJ: Prentice-Hall, 1991.
35. Boyd S., El Ghaoui L., Feron E., Balakrishnan V. Linear Matrix Inequalities in System and Control Theory. Philadelphia: SIAM, 1994.
36. Boyd S., El Ghaoui L. Method of centers for minimizing generalized eigenvalues // Linear Algebra and Its Applications. 1993. V. 188. P. 63–111.
37. Boyd S., Vandenberghe L. Convex Optimization. Cambridge: Cambridge Univ. Press, 2004. (Имеется электронный вариант <http://www.stanford.edu/~boyd/>.)
38. de Klerk E. Aspects of Semidefinite Programming: Interior Point Algorithms and Selected Applications. Boston, Dordrecht, London: Kluwer Academic Publishers, 2002.
39. de Oliveira M.C., de Farias D.P., Geromel J.C. LMISol (Version 1.0). User's Guide. April, 1997.
<http://www.dt.fee.unicamp.br/~mauricio/lmisol10.html>.

40. *Gahinet P., Nemirovski A., Laub A.J., Chilali M.* The LMI Control Toolbox. For Use with Matlab. User's Guide. Natick, MA: The MathWorks, Inc., 1995.
41. *Hansson A., Wallin R., Vandenberghe L.* Efficiently solving semidefinite programs originating from the KYP lemma using standard primal-dual solvers. Technical report LiTH-ISY-R-2503, Linköpings universitet, Linköping, 2003. <http://www.control.isy.liu.se>.
42. *Henrion D., Kocvara M., Stingl M.* Solving simultaneous stabilization BMI problems with PENNON. LAAS-CNRS Research Report No. 03549, December 2003. <http://www.laas.fr/~henrion/publis.html>.
43. *Henrion D., Lasserre J.-B.* GloptiPoly: Global Optimization over Polynomials with Matlab and SeDuMi. Version 2.0. April 29, 2002. <http://www.laas.fr/~henrion/software/gloptipoly/>.
44. *Kalman R.E.* Lyapunov functions for the problem of Lur'e in automatic control // Proc. Nat. Acad. Sci. USA. 1963. V. 49. № 2. P. 201–205.
45. *Kao C.-Y., Megretski A., Jönsson U.* Specialized fast algorithms for IQC feasibility and optimization problems // Automatica. 2004. V. 40. № 2. P. 239–252.
46. *Kao C.-Y.* Efficient Computational Methods for Robustness Analysis. D.Sc. Thesis. Massachusetts Inst. of Technology. 2002.
47. *Kočvara M., Stingl M.* PENBMI User's Guide (Version 1.1). PENOPT Gbr., November 2003. <http://www.penopt.com>.
48. *Kočvara M., Stingl M.* PENSMP User's Guide (Version 1.1). PENOPT Gbr., November 2003. <http://www.penopt.com>.
49. *Liao X., Chen G., Sanches E.N.* LMI-based approach for asymptotically stability analysis of delayed neural networks // IEEE Trans. Circuits and Systems—I: Fundamental Theory and Applications. 2002. V. 49. № 7. P. 1033–1039.
50. *Lobo M., Vandenberghe L., Boyd S., Lebret H.* Applications of second-order cone programming // Linear Algebra and Its Applications. 1998. V. 284. P. 193–228.
51. *Löfberg J.* YALMIP manual. Version 3. 2004. <http://control.ee.ethz.ch/~joloef/yalmip.mssql>.

52. Megretski A., Kao C.-Y., Jönsson U., Rantzer A. A Guide to IQC β : Software for Robustness Analysis. Feb. 18, 2003. <http://www.math.kth.se/~cykao/>.
53. Nesterov Y.E., Todd M. Self-scaled barriers and interior-point methods for convex programming // Mathematics of Operations Research. 1997. V. 22. P. 1–42.
54. Nesterov Yu., Nemirovski A. Interior-Point Polynomial Algorithms in Convex Programming. Philadelphia: SIAM, 1994.
55. Nikoukhah R., Delebecque F., El Ghaoui L. LMITOOL: A Package for LMI Optimization in Scilab. User's Guide. <ftp://ftp.ensta.fr/pub/elghaoui/lmitool>.
56. Peaucelle D., Henrion D., Labit Y., Taitz K. User's Guide for SEDUMI INTERFACE 1.04. 13th September, 2002. <http://www.laas.fr/~peaucell/SeDuMiInt.html>.
57. PolyX. The Polynomial Toolbox for Matlab. Manual. Version 2.0. March, 1999. <http://www.polyx.cz>.
58. Prajna S., Papachristodoulou A., Parrilo P.A. Sum of Squares Optimization Toolbox for MATLAB. User's Guide. Version 1.00. April 11, 2002. <http://www.cds.caltech.edu/sostools/>.
59. Pyatnitskiy Ye.S., Skorodinskiy V.I. Numerical methods of Lyapunov function construction and their application to the absolute stability problem // Systems Control Lett. 1982. V. 2. № 2. P. 130–135.
60. Renegar J. A Mathematical View of Interior-Point Methods in Convex Optimization. Philadelphia: SIAM, 2001.
61. Scherer C., Weiland S. Linear Matrix Inequalities in Control. Version 3.0. October, 2000. <http://www.cs.ele.tue.nl/SWeiland/lmi.htm>.
62. Sturm J.F. Using SeDuMi 1.02, a MATLAB Toolbox for Optimization over Symmetric Cones (Updated for Version 1.05). August, 1998 – October, 2001. <http://fewcal.kub.nl/~sturm>.
63. Terlaki T. (Ed.). Interior-Point Methods of Mathematical Programming. Dordrecht: Kluwer Academic Publishers, 1996.
64. Vandenberghe L., Boyd S. Semidefinite programming // SIAM Review. 1996. V. 38. № 1. P. 49–95.

65. *Vandenberghe L., Boyd S.* SP: Software for Semidefinite Programming. User's Guide. Version 1.0. November 1998.
<http://www.ee.ucla.edu/~vandenbe/sp.html>.
66. *Wallin R.* User Manual for `kypd_solver`. January 26, 2004.
<http://www.control.isy.liu.se/>.
67. *Wright S.J.* Primal-Dual Interior-Point Methods. Philadelphia: SIAM, 1997.
68. *Wu S.-P., Boyd S.* SDPSOL. A Parser/Solver for Semidefinite Programming and Determinant Maximization Problems with Matrix Structure. User's Guide. Version Beta. May 31, 1996.
<http://www.stanford.edu/~boyd/SDPSOL/>.
69. *Ye Y.* Interior-Point Algorithms: Theory and Analysis. New York: John Wiley & Sons, 1997.

Указатель команд и функций

KYPD

kypd, 134
kypd solver, 134

LMILab

basiclmi, 57
dec2mat, 49
decinfo, 48
decnbr, 45
defcx, 50
dellmi, 58
delmvar, 58
evallmi, 57
feasp, 53
getlmis, 42
gevp, 55
lmiedit, 42
lmiinfo, 45
lminbr, 45
lmiterm, 39
lmivar, 36
mat2dec, 50
matnbr, 45
mincx, 54
newlmi, 39
setlmis, 35
setmvar, 59
showlmi, 58

LMITOOL

lmisolver, 69
lmitool, 70, 71

MATLAB

lqr, 131
num2str, 88

SeDuMi Interface

get, 87, 102
sdmeq, 96
sdmget, 108
sdmineq, 91
sdmlme, 95
sdmlmi, 89
sdmobj, 98
sdmpb, 83
sdmset, 106
sdmsol, 99
sdmvar, 84

YALMIP

binvar, 119
checkset, 125
double, 119, 125
dual, 125
fixsdpvar, 119
intvar, 119
lmi, 117
sdpsettings, 123
sdpvar, 118
set, 117, 120
setsdpvar, 119
solvesdp, 123
yalmip, 132
yalmipdemo, 132
yalmiptest, 132

А. Н. Чурилов, А. В. Гессен

ИССЛЕДОВАНИЕ ЛИНЕЙНЫХ МАТРИЧНЫХ НЕРАВЕНСТВ

ПУТЕВОДИТЕЛЬ ПО ПРОГРАММНЫМ ПАКЕТАМ

Книга содержит описание программных средств исследования линейных матричных неравенств. Подробно рассматриваются программные пакеты LMILab, LMITOOL, SeDuMi Interface, YALMIP и KYPD, которые являются пакетами расширения систем программирования MATLAB и Scilab



9 785288 035852 >