Формулы (5.5.9), (5.5.I0), (5.5.I6) и (5.5.I8) позволяют, в частности, вести рекуррентное вычисление векторов $c^{(k)}$ - последних столбцов матриц $A_k^{-1}$, которые и нужны для реализации метода окаймления. В случае, когда блоки $x_i$, $y_i$ суть скалярные матрицы, построенный нами алгоритм упрощается и может быть записан следующим образом:

$k = 0$:

$$u_{l0}^{(0)} = a_{00}^{-1}\psi_{l0}; \quad v_{l0}^{(0)} = \psi_{l0}a_{00}^{-1}, \quad l = 1, \dots, r,$$

$$z_0^{(0)} = a_{00}^{-1}b_0;$$

$k = 1, \dots, n-1$:

$$\alpha_{lk} = \psi_{lk} - \sum_{j=0}^{k-1} a_{kj}u_{lj}^{(k-1)}, \quad l = 1, \dots, r,$$

$$\beta_{lk} = \psi_{lk} - \sum_{j=0}^{k-1} v_{lj}^{(k-1)}a_{jk}, \quad l = 1, \dots, r,$$

$$\gamma_k = b_k - \sum_{j=0}^{k-1} a_{kj}z_j^{(k-1)},$$

$$c_k^{(k)} = s_k^{(k)} = (x_k - y_k)^{-1}\sum_{l=1}^{r}\alpha_{lk}\beta_{lk}, \qquad (5.5.19)$$

$$c_i^{(k)} = (y_k - y_i)^{-1}\sum_{l=1}^{r}u_{li}^{(k-1)}\beta_{lk}c_k^{(k)}, \quad i = 0, 1, \dots, k-1,$$

$$s_j^{(k)} = (x_j - x_k)^{-1}c_k^{(k)}\sum_{l=1}^{r}\alpha_{lk}v_{lj}^{(k-1)}, \quad j = 0, 1, \dots, k-1,$$

$$u_l^{(k)} = 1\,u_l^{(k-1)} + c^{(k)}\alpha_{lk}, \quad l = 1, \dots, r,$$

$$v_l^{(k)\prime} = \left(1\,v_l^{(k-1)}\right)' + \beta_{lk}s_l^{(k)\prime}, \quad l = 1, \dots, r,$$

$$z^{(k)} = 1\,z^{(k-1)} + c^{(k)}\gamma_k.$$



При подсчете числа операций следует учесть, что $a_{ij}$ и $a_{jk}$ также вычисляются согласно (5.5.5). При $p = 1$ общее число операций умножения-деления составляет $(4r + 1)n^2$ (в главном члене). Число сложений-вычитаний такое же.

## § 6. Методы векторизации

### 6.I. Последовательные и векторные алгоритмы

Пусть фиксирован конечный запас типов операций с конечным числом входов и выходов. Тогда последовательный алгоритм будем считать заданным, если каким-либо образом определена последовательность операций из имеющегося запаса и отвечающих им операций.

Предположим, что множество операций в алгоритме разбито на непересекающиеся подмножества $M_1, \dots, M_h$, и при этом если в $M_j$ используется какой-либо результат, полученный в $M_i$, то $i \le j$. Такое разбиение называется обобщенной параллельной формой алгоритма и просто параллельной формой, если для всех $i$ внутри $M_i$ операции не обмениваются результатами. Подмножества $M_i$ называются ярусами, а их число $h$ - высотой обобщенной параллельной формы.

В действительности обычно рассматривается некоторое семейство последовательностей и отвечающих им разбиений. Предположим, что для каждой из них любой из ярусов имеет параллельную форму, высота которой, вообще говоря, зависит от особенностей рассматриваемого семейства, но не может зависеть от характеристик конкретной последовательности, например от числа операций в ней или в ее ярусах. Введем укрупненные операции соответственно ярусам $M_1, \dots, M_h$. Для каждой из них входные и выходные данные представим в виде совокупностей векторов, число которых не должно зависеть от их размерностей, и, более того, предположим, что для всего рассматриваемого семейства последовательностей удается выделить лишь несколько типов таких укрупненных операций - будем называть их векторными операциями. Алгоритм, представленный как последовательность векторных операций, будем называть векторным алгоритмом, а процесс его получения - векторизацией.

С точки зрения максимального распараллеливания следует добиваться наименьшей высоты параллельной формы. Однако обычно стремятся не к максимальной, а к некоторой "разумной" параллельности, обеспечивающей определенные структурные качества. Поэтому в последнее время усиленное внимание уделяется разработке векторных алгоритмов,

8957

вполне соответствующих потребностям широкого класса вычислительных систем: векторно-конвейерных, систолических и др. Обычно считается, что данные, составляющие один вектор, однотипны. В устройствах памяти ЭВМ они размещаются особым образом, упрощающим их запись и считывание. Подразумевается также однотипность их обработки.

Все исследования методов преобразования алгоритмов и программ в принципиальном плане можно отнести к одному из двух направлений по следующему признаку:

- исходный алгоритм сохраняется, но приобретает новую форму записи, в которой явно указывается присущий ему параллелизм;
- исходный алгоритм каким-либо образом изменяется.

Обычно именно первое направление связывается с понятием распараллеливания. Во многих случаях достаточно того параллелизма, который неявно присутствует в уже имеющихся алгоритмах и программах. Поэтому первое направление интенсивно развивается, причем по пути создания общей методологии и реализующих ее инструментальных систем.

В то же время есть много примеров, когда имеющегося параллелизма недостаточно. Таковы многие алгоритмы оптимизации, решения нелинейных уравнений. Сюда же относятся различные рекуррентные алгоритмы для алгебраических задач с различного рода спецификой (тёплицевость, ленточность и т.д.). Во втором направлении получено много частных результатов и почти ничего, относящегося к достаточно широким классам алгоритмов. Исключение составляют лишь схема сдваивания, рекуррентное сдваивание и методика рекурсивного расщепления задачи.

Если известно, что алгоритм не распараллеливается, то единственная возможность его ускорить связана с переходом к новому алгоритму. Учитывая трудность задачи, было бы заманчиво иметь какие-либо общие подходы к преобразованию алгоритмов. Ниже описывается общий метод векторизации, применяемый к последовательным алгоритмам с некоторыми общими структурными особенностями. В § 8 с его помощью на базе алгоритмов из § 5 будут построены конкретные векторные алгоритмы для матриц типа тёплицевых. Подчеркнем, что это будут новые математические алгоритмы, т.е. исследования относятся именно ко второму направлению.

## 6.2. Как строить векторные алгоритмы

Рассмотрим следующий класс алгоритмов, интересуясь сейчас не смыслом решаемой задачи, а лишь структурой предписаний. Будем считать, что в алгоритме имеется нестандартный начальный шаг и последовательность $N$ однотипных шагов.



На начальном шаге каким-либо способом определяются векторы $q_1^{(0)},\dots,q_r^{(0)}$ размерности $n$ и скалярные величины, рассматриваемые как компоненты вектора $p^{(0)}$. Пусть эти действия выполняются с максимальной параллельностью - здесь и в дальнейшем это будет означать, что высота соответствующей параллельной формы не зависит от $n$ или $N$.

Далее на $k$-м шаге $(1 \le k \le N)$ происходит переход от $q_1^{(k-1)},\dots,q_r^{(k-1)}, p^{(k-1)}$ к $q_1^{(k)},\dots,q_r^{(k)}, p^{(k)}$. Для этого:

1) вычисляются линейные формы $\varphi_s^{(k)} = l_s^{(k)T} q_{v_s}^{(k-1)}$ от компонент одного или нескольких векторов $q_j^{(k-1)}$; $j = v_s$;

2) вычисляются скалярные величины

$$\gamma_{ij}^{(k)} = \gamma_{ij}^{(k)}\left(\varphi_1^{(k)},\dots,\varphi_m^{(k)}\right), \quad 1 \le i, j \le n;$$

3) вычисляются векторы $q_i^{(k)}$ по формулам

$$q_i^{(k)} = \sum_{j=1}^{n} P_{ij}^{(k)} q_j^{(k-1)} \gamma_{ij}^{(k)}, \quad 1 \le i \le n.$$

Предполагается, что векторы $l_i$ и матрицы $P_{ij}^{(k)}$ известны заранее или с максимальной параллельностью вычисляются на начальном шаге. Действие 2) рассматривается как "черный ящик", однако будем считать, что оно выполняется с максимальной параллельностью. Любой такой алгоритм будем называть алгоритмом $T$-типа, если для всех $i, k$ число отличных от нуля матриц $P_{ij}^{(k)}$ $(1 \le j \le n)$ не зависит от $n$ или $N$. Будем считать, что если $P_{ij}^{(k)} = 0$, то и $\gamma_{ij}^{(k)} = 0$. В алгоритме $T$-типа формально допускается зависимость $n$ от $n$ или $N$, но в любом случае действие 3) выполняется с максимальной параллельностью. Если в алгоритме $T$-типа для всех $i, j, k$ матрица $P_{ij}^{(k)}$ умножается на вектор с максимальной параллельностью, то будем называть его просто $T$-алгоритмом.

Любой $T$-алгоритм, очевидно, имеет параллельную форму высотой $O(nN)$. Поскольку действие 2) рассматривается как "черный ящик", число ярусов $O(N)$ нужно считать нижней границей; приблизиться к ней мешает только одно - вычисление линейных форм в п. 1). Естественно вспомнить о схеме сдваивания - с ее помощью число ярусов очевидно снижается до $O(N\log_2 n)$. Однако нижняя граница при этом, как видим, не достигается. Таким образом, требуется какой-то иной подход к получению параллельного алгоритма.

Теорема 6.2.I. Для любого $T$-алгоритма существует алгоритм, который решает ту же задачу и при этом его параллельная форма имеет высоту $O(N + \log_2 N \log_2 n)$.

Доказательство. Рассмотрим $k$-й шаг $T$-алгоритма. Возьмем любую из вычисляемых на этом шаге линейных форм $\varphi_s^{(k)} = \ell^T q_{\nu_s}^{(k-1)}$, $\ell = \ell_s^{(k)}$ и наряду с ней рассмотрим следующие линейные формы:

$$\psi_{i_1 j_1}^{(k-1)} = \left(\ell^T P_{i_1 j_1}^{(k-1)}\right) q_{j_1}^{(k-2)}, \quad 1 \le j_1 \le n; \ i_1 = \nu_s. \qquad (6.2.I)$$

Тогда

$$\varphi_s^{(k)} = \sum_{j_1=1}^{n} \psi_{i_1 j_1}^{(k-1)} \gamma_{i_1 j_1}^{(k-1)}, \qquad (6.2.2)$$

причем согласно определению $T$-алгоритма число ненулевых слагаемых не зависит от $n$ или $N$. Поэтому формула (6.2.2) реализуется с максимальной параллельностью. Но для этого, конечно, сначала необходимо вычислить линейные формы $\psi_{i_1 j_1}^{(k-1)}$. Чтобы их получить, поступим аналогичным образом, т.е. введем новые линейные формы

$$\psi_{i_1 j_1; i_2 j_2}^{(k-2)} = \left(\ell^T P_{i_1 j_1} P_{i_2 j_2}\right) q_{j_2}^{(k-3)}. \qquad (6.2.3)$$

Тогда

$$\psi_{i_1 j_1}^{(k-2)} = \sum_{j_2=1}^{n} \psi_{i_1 j_1; j_1 j_2}^{(k-2)} \gamma_{j_1 j_2}^{(k-2)}, \qquad (6.2.4)$$

причем и эта формула реализуется с максимальной параллельностью. Чтобы получить $\psi_{i_1 j_1; i_2 j_2}^{(k-2)}$, вводим новые линейные формы $\psi_{i_1 j_1; i_2 j_2; i_3 j_3}^{(k-3)}$ и т.д. В результате на начальном шаге потребуется получить линейные формы

$$\psi_{i_1 j_1; \ldots; i_{k-1} j_{k-1}; j_k}^{(0)} = \left(\ell^T P_{i_1 j_1}^{(k-1)} \ldots P_{i_{k-1} j_{k-1}}^{(1)}\right) q_{j_k}^{(0)}. \qquad (6.2.5)$$

Для вычисления векторов $\ell^T P_{i_1 j_1} \ldots P_{i_{k-1} j_{k-1}}$ воспользуемся схемой сдваивания - тогда потребуется $O(\log N \log n^{k-1})$ параллельных шагов. Все остальные действия, связанные с получением дополнительно введенных линейных форм, реализуются посредством $O(k)$ параллельных шагов. Вычисления, отвечающие различным векторам $\ell = \ell_s^{(k)}$, могут быть выполнены параллельно. Таким образом, все линейные формы могут быть получены за $O(N + \log N \log n)$ параллельных шагов. После этого уже ничто не мешает найти векторы $q_{\nu_1}^{(k)}, \ldots, q_{\nu_n}^{(k)}$ с затратой $O(N)$ параллельных шагов. Теорема доказана.



Замечание I. Если отказаться от применения схемы сдваивания, то от $T$-алгоритма можно придти к параллельному алгоритму с числом шагов $O(N + n)$, если для всех $i, j$ умножение на вектор матрицы $P_{ij}^{(k)T}$ выполняется с максимальной параллельностью. Последнее в общем случае не вытекает из аналогичного свойства для матриц $P_{ij}^{(k)}$.

Замечание 2. Утверждение теоремы 6.2.I справедливо по отношению к любому алгоритму $T$-типа, в котором $n$ не зависит от $n$ или $N$.

В соответствующем параллельном алгоритме линейные формы вычисляются точно так же, как показано в доказательстве теоремы 6.2.I. После того как они найдены, векторы $q_{\nu_1}^{(k)}, \ldots, q_{\nu_n}^{(k)}$ вычисляются в согласии с формулой

$$\begin{bmatrix} q_{\nu_1}^{(k)} \\ \ldots \\ q_{\nu_n}^{(k)} \end{bmatrix} = \begin{bmatrix} \gamma_{11}^{(k)} P_{11}^{(k)} & \ldots & \gamma_{1n}^{(k)} P_{1n}^{(k)} \\ \ldots & \ldots & \ldots \\ \gamma_{n1}^{(k)} P_{n1}^{(k)} & \ldots & \gamma_{nn}^{(k)} P_{nn}^{(k)} \end{bmatrix} \ldots \begin{bmatrix} \gamma_{11}^{(1)} P_{11}^{(1)} & \ldots & \gamma_{1n}^{(1)} P_{1n}^{(1)} \\ \ldots & \ldots & \ldots \\ \gamma_{n1}^{(1)} P_{n1}^{(1)} & \ldots & \gamma_{nn}^{(1)} P_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} q_{\nu_1}^{(0)} \\ \ldots \\ q_{\nu_n}^{(0)} \end{bmatrix}, \qquad (6.2.6)$$

которая реализуется с использованием схемы рекуррентного сдваивания. Можно сформулировать и более сильное утверждение, освободившись от ограничения на $n$.

Как видим, параллельные алгоритмы строятся по следующему плану: вводится какое-то число дополнительно вычисляемых линейных форм, для которых на каждом шаге организуется рекуррентный пересчет, обладающий высокой параллельностью; при этом сильно укрупняется начальный шаг, но все дополнительные вычисления хорошо распараллеливаются. Главный недостаток - это очень большое число новых линейных форм. Чтобы построить практически полезный параллельный алгоритм, необходимо добиться существенного сокращения этого числа. Это и будет предметом наших ближайших исследований.

Теорема 6.2.2. Пусть в $T$-алгоритме все ненулевые матрицы $P_{ij}^{(k)}$ суть единичные. Тогда существует векторный алгоритм, решающий ту же

задачу, с параллельной формой, имеющей $O(N + r)$ ярусов, и при этом требуется запоминать дополнительно $r$ векторов одинаковой размерности, равной общему числу линейных форм на всех шагах данного $T$-алгоритма.

Доказательство. Каким-либо способом перенумеруем линейные формы, встречающиеся на всех шагах данного $T$-алгоритма, и предположим, что согласно принятой нумерации они задаются последовательностью вектор-столбцов $l_1, \dots, l_m$. Составим матрицу $L$ размеров $M \times r$, считая что при $1 \le i \le M$ ее $i$-я строка есть $l_i^T$. Введем $r$ семейств $M$-мерных векторов

$$d_i^{(k)} = L q_i^{(k)}, \quad 1 \le i \le r. \qquad (6.2.7)$$

Принимая во внимание структуру $T$-алгоритма, находим

$$d_i^{(k)} = \sum_{j=1}^{r} d_j^{(k-1)} \gamma_{ij}^{(k)}, \quad 1 \le i \le r. \qquad (6.2.8)$$

Число отличных от нуля слагаемых не зависит от $N$ или $r$, поэтому формулы (6.2.8) реализуются с максимальной параллельностью. Чтобы начать эти вычисления, необходимо иметь векторы $d_i^{(0)}$ $(1 \le i \le r)$. Получить их можно, используя (6.2.7), - это потребует $O(r)$ параллельных шагов. Векторы $q_{i_1}^{(k)}, \dots, q_{i_r}^{(k)}$ вычисляются в соответствии с инструкциями исходного $T$-алгоритма. Поскольку для каждого $k$ п. 3) реализуется с максимальной параллельностью, для этого потребуется всего $O(N)$ параллельных шагов. Теорема доказана.

Замечание I. В построенном векторном алгоритме не использовалась схема сдваивания. Ее можно было бы применить для вычисления $d_i^{(0)}$ $(1 \le i \le r)$; тогда мы получили бы параллельный алгоритм с числом параллельных шагов $O(N + \log r)$, но с несколько большим числом дополнительно запоминаемых величин.

Замечание 2. При вычислении векторов $d_i^{(k)}$ $(1 \le i \le r)$ в векторном алгоритме, отвечающем доказательству теоремы 6.2.2, определяются, в частности, и те их компоненты, которые соответствуют линейным формам, находящимся на всех шагах $T$-алгоритма, предшествующих $k$-му. В действительности эти компоненты нигде не используются, поэтому векторный алгоритм можно упростить, исключив "лишние" операции. Обратим внимание на то, что в исходном $T$-алгоритме линейные формы на каждом шаге вычисляются с одними и теми же затратами арифметических операций, в то время как после векторизации эти затраты умень-



шаются от шага к шагу.

Теорема 6.2.3. Пусть имеется $T$-алгоритм, и $M \times r$-матрица $L$ такова, что для любой линейной формы определяющая ее вектор-строка $l_s^T$ совпадает с некоторой строкой в $L$. Предположим, что существуют $M \times M$-матрицы $Q_{ij}^{(k)}$, удовлетворяющие соотношениям

$$Q_{ij}^{(k)} L = L P_{ij}^{(k)}, \quad 1 \le i, j \le r, \quad 1 \le k \le N, \qquad (6.2.9)$$

и такие, что любая из них умножается на вектор с максимальной параллельностью. Тогда существует векторный алгоритм, решающий ту же задачу, с параллельной формой, имеющей $O(N + r)$ ярусов, и при этом требуется запоминать дополнительно $r$ векторов размерности $M$.

Доказательство. Определим вектор-столбцы $d_i^{(k)}$ $(1 \le i \le r)$ в соответствии с (6.2.7). Тогда, принимая во внимание (6.2.9), находим

$$d_i^{(k)} = \sum_{j=1}^{r} Q_{ij}^{(k)} d_j^{(k-1)} \gamma_{ij}^{(k)}. \qquad (6.2.10)$$

Остается лишь учесть, что число ненулевых слагаемых не зависит от $N$ или $r$ в силу определения $T$-алгоритма, а $Q_{ij}^{(k)}$ умножается на вектор с максимальной параллельностью по условию теоремы. Поэтому, используя (6.2.10), получаем алгоритм с числом параллельных шагов $O(N + r)$. Теорема доказана.

Теорема 6.2.4. Пусть задан $T$-алгоритм, в нем на каждом шаге вычисляется только одна линейная форма, и все они порождаются общей вектор-строкой $l^T$. Предположим, что матрицы $P_{ij}^{(k)}$ одинаковы для всех $k$, т.е. $P_{ij}^{(k)} \equiv P_{ij}$, и пусть множество всех ненулевых матриц, которые можно получить с помощью операции умножения матриц, исходя из $P_{ij}$ $(1 \le i, j \le r)$, конечно и содержит $M$ матриц. Тогда существует векторный алгоритм, решающий ту же задачу, с параллельной формой, имеющей высоту $O(N + r)$, и при этом требуется запоминать дополнительно $r$ векторов размерности $M$.

Доказательство. Обозначим $F_1, \dots, F_M$ матрицы, исчерпывающие множество матриц, получаемых с помощью операции умножения и взятых в любом порядке и числе матриц $P_{ij}$ $(1 \le i, j \le r)$. Пусть $L$ есть $M \times r$-матрица, составленная из строк $l^T F_1, \dots, l^T F_M$. Фиксируем $i, j$ и рассмотрим матрицу $LP_{ij}$. Очевидно, ее строки имеют вид $l^T(F_1 P_{ij}), \dots, l^T(F_M P_{ij})$. По условию теоремы любая

матрица, заключенная в скобки, есть не что иное, как одна из матриц $F_1, \dots, F_M$. Следовательно, матрица $LP_{ij}$ имеет строки, совпадающие с какими-то строками матрицы $L$ либо нулевые, т.е. $LP_{ij} = Q_{ij}L$, где $Q_{ij}$ в каждой строке имеет только нули и, возможно, одну единицу. Понятно, что умножение матрицы $Q_{ij}$ на вектор выполняется с максимальной параллельностью. Вводя дополнительные линейные формы, определяемые строками матрицы $Q_{ij}$, мы получаем $T$-алгоритм, удовлетворяющий условиям теоремы 6.2.3! Значит, векторный алгоритм с требуемыми свойствами существует. Теорема доказана.

Свойство алгоритма быть $T$-алгоритмом обычно обнаруживается без особого труда. Однако конкретный вид векторов $q_i^{(k)}$ и линейных форм от их компонент определяется неоднозначно. В нашем методе векторизации мы отделяем вычисление линейных форм от вычисления векторов $q_i^{(k)}$. С учетом этого бывает полезно провести расщепление множества линейных форм на классы и строить параллельные алгоритмы для каждого из классов в отдельности. В ряде случаев удается получить классы, позволяющие применить теоремы 6.2.2-6.2.4.

## § 7. Векторные алгоритмы для матриц типа тёплицевых

### 7.I. Тёплицевы матрицы

Рассмотрим алгоритм (5.2.5). Его максимальная параллельная форма содержит $O(n^2)$ ярусов, т.е. алгоритм практически не распараллеливается. Однако очевидно, что это есть $T$-алгоритм. Множество всех линейных форм естественным образом разбивается на два класса: $\{F_k\}$ и $\{G_k\}$.

Введем следующие $n-1$-мерные векторы:

$$q_1^{(k)} = [0 \dots 0\ x_0^{(k)} \dots x_k^{(k)}]', \quad q_2^{(k)} = [0 \dots 0\ y_0^{(k)} \dots y_k^{(k)}]', \tag{7.I.1}$$
$$q_3^{(k)} = [0 \dots 0\ x_k^{(k)} \dots x_0^{(k)}]', \quad q_4^{(k)} = [0 \dots 0\ y_k^{(k)} \dots y_0^{(k)}]'.$$

Тогда

$$F_k = l_1' q_1^{(k-1)}, \quad l_1 = [a_{n-1} \dots a_1]', \tag{7.I.2}$$



$$G_k = l_2' q_4^{(k-1)}, \quad l_2 \equiv [a_{-n+1} \dots a_{-1}]'. \tag{7.I.3}$$

Что же касается действий п. 3) $T$-алгоритма, то они выглядят следующим образом:

$$q_1^{(k)} = Z q_1^{(k-1)} + q_2^{(k-1)} s_k, \quad q_2^{(k)} = Z q_1^{(k-1)} t_k + q_2^{(k-1)}; \tag{7.I.4}$$

$$q_3^{(k)} = q_3^{(k-1)} + Z q_4^{(k-1)} s_k, \quad q_4^{(k)} = q_3^{(k-1)} t_k + Z q_4^{(k-1)}; \tag{7.I.5}$$

$$Z = \begin{bmatrix} 0 & I & & 0 \\ & 0 & I & \\ & & \dots & \\ 0 & & & 0\ I \end{bmatrix}_{(n-1)\times(n-1)} \tag{7.I.6}$$

Исходный $T$-алгоритм очевидно распадается на два $T$-алгоритма: один - связанный с (7.I.2), (7.I.4) и другой - связанный с (7.I.3), (7.I.5). Каждый из этих двух $T$-алгоритмов удовлетворяет условиям теоремы 6.2.4. Здесь роль матриц $P_{ij}$ играет $I$ и $Z$. Множество матриц, которые с помощью операции умножения порождаются матрицами $I$ и $Z$, составляет множество целых неотрицательных степеней матрицы $Z$. Оно конечно, так как $Z^{n-1} = 0$, и содержит $M = n - 1$ ненулевых матриц. Согласно доказательству теоремы 6.2.4 получаем

$$L_1 = \begin{bmatrix} l_1' \\ l_1' Z \\ \dots \\ l_1' Z^{n-3} \\ l_1' Z^{n-2} \end{bmatrix} = \begin{bmatrix} a_{n-1} & a_{n-2} & \dots & \dots & a_1 \\ & a_{n-1} & a_{n-2} & \dots & a_2 \\ & & & \dots & \dots \\ & 0 & & a_{n-1} & a_{n-2} \\ & & & & a_{n-1} \end{bmatrix}, \tag{7.I.7}$$

8957
I9-2