УДК 519.688

О БЫСТРОМ ОБОБЩЕННОМ РАЗЛОЖЕНИИ БРЮА В ОБЛАСТИ

© **Г. И. Малашонок**

Ключевые слова: обобщенное разложение Брюа, рекурсивный блочный алгоритм, коммутативная область.

Рассматривается детерминистский рекурсивный алгоритм вычисления обобщенного разложения Брюа для матриц над коммутативными областями. По числу кольцевых операций алгоритм имеет такую же сложность, как и матричное умножение. Алгоритм входит в библиотеку алгоритмов системы Mathpar.

1 Введение

В линейной алгебре важную роль играет LU-разложение матриц. Такое разложение можно вычислить с помощью метода последовательного исключения Гаусса. В общем случае могут встретиться нулевые миноры в верхнем левом углу данной матрицы. Это приведет к тому, что появятся нулевые ведущие элементы. В этом случае вычисляют более общий вид разложения $PAQ = LU$, где P и Q — матрицы перестановок.

Рене Брюа (Francois Georges René Bruhat) одним из первых стал применять другое треугольное разложение. Он использовал разложение матрицы в виде $A = VwU$, где V и U являются невырожденными верхними треугольными матрицами и w — это матрица перестановок. Такое разложение играет важную роль в теории групп. Обобщенное разложение Брюа было впервые рассмотрено в работе Д. Григорьева [1]. Он рассматривал вырожденные верхние треугольные матрицы V и U. В работе [2] исследовалось разложения Брюа для разреженных невырожденных матриц над полем и было показано, что треугольные сомножители сохраняют разреженность гораздо лучше, чем в случае LU-разложения.

Быстрое матричное умножение и быстрое блочное матричное обращение были предложены Штрассеном [3]. Он показал, что алгоритм обращения матриц имеет такую же сложность, как и алгоритм умножения матриц. Однако алгоритм обращения Штрассена нельзя применять для матриц общего вида, так как он требует, чтобы ведущие миноры матрицы не равнялись нулю.

Позже были получены рекурсивные алгоритмы для вычисления присоединенной и обратной матрицы в коммутативной области со сложностью матричного умножения [4]-[7].

В общем случае, чтобы получить быстрый алгоритм обращения, необходимо каждый раз искать подходящий ненулевой элемент и переставлять его на место ведущего элемента путем перестановок строк и столбцов. Банч и Хопкрофт опубликовали такой алгоритм обращения матрицы (Bunch and Hopkroft [8]).

Параллельные матричные алгоритмы исследовались в работах [9] – [11]. Так в работе [10] получен быстрый рекурсивный алгоритм для присоединенной и обратной матрицы, а также ступенчатой матрицы, определителя и ядра оператора для матриц общего вида над коммутативной областью.

В работе [12] предложен алгоритм для вычисления быстрого рекурсивного обобщенного разложения Брюа для матриц над полем произвольной характеристики, который имеет сложность матричного умножения.

Мы продолжаем исследования, направленные на разработку быстрых рекурсивных детерминистских блочных алгоритмов в коммутативных областях, в духе работ [9] – [12].

В настоящей работе предлагается алгоритм вычисления обобщенного разложения Брюа для матриц общего вида над коммутативной областью. Это блочно-рекурсивный алгоритм, который имеет сложность матричного умножения. Вычисляется разложение матрицы $A = LdU$, где матрица L — нижняя треугольная, U — верхняя треугольная, d — матрица перестановок, домноженная на обратную диагональную матрицу. Каждая из матриц, стоящих в правой части, имеет такой же ранг, что и матрица A.

Если нужно получить разложение Брюа $A = VdU$, в котором матрица V является верхней треугольной, то можно воспользоваться матрицей перестановок I'. Эта матрица меняет местами равноудаленные от концов элементы вектора, у нее единичные элементы стоят на неглавной диагонали. Найдем сначала разложение $I'A = LdU$, а затем найдем разложение Брюа: $A = (I'LI')(I'd)U$.

2 Основные определения

Мы введем некоторые обозначения, которые будут использоваться в следующих разделах.

Пусть R — коммутативное кольцо, $A \in R^{n\times m}$ — матрица размера $n \times m$ над кольцом R. Будем пользоваться следующими обозначениями для подматриц.
$A[^{i_1,\ldots,i_r}_{j_1,\ldots,j_s}]$ — подматрица, стоящая на пересечении строк $i_1, \ldots, i_r$ и столбцов $j_1, \ldots, j_c$ матрицы A; $A^k_{i,j} = A[^{1,\ldots,k-1,i}_{1,\ldots,k-1,j}]$ — подматрица порядка k, полученная окаймлением верхней левой угловой подматрицы порядка $k-1$ строкой i и столбцом j.

Введем обозначения для миноров. Будем обозначать миноры малыми греческими буквами, при этом верхний индекс будет всегда обозначать порядок минора: $\alpha^k_{i,j} = \det(A^k_{i,j})$, $\alpha^k = \alpha^k_{k,k}$; $\delta^k_{i,j} = \det(A[^{1,2,\ldots,k}_{1,\ldots,i-1,j,i+1,\ldots,k}])$ — верхний левый угловой минор порядка k после замены столбца i на столбец j в матрице A, $1 \leqslant k \leqslant \min(n,m)$, $i,j \leqslant m$; $\sigma^k_{i,j} = \det(A[^{1,\ldots,j-1,i,j+1,\ldots,k}_{1,2,\ldots,k}])$ — верхний левый угловой минор порядка k после замены строки j на строку i в матрице A, $1 \leqslant k \leqslant \min(n,m)$, $i,j \leqslant n$; $\beta^k_{j,i}$ — алгебраическое дополнение элемента (i,j) в матрице $A^{0,k+1}_{0,k+1}$.

Пусть $l = \min(n,m)$. Для $k = 1, 2, \ldots, l$ определим матрицы миноров матрицы A:

$$\mathcal{U} = \mathcal{U}(A) = (\alpha^i_{i,j}) \in R^{m\times m},\ \mathcal{L} = \mathcal{L}(A) = (\alpha^j_{i,j}) \in R^{n\times n},$$

$$\mathcal{G}^k = \mathcal{G}^k(A) = (\delta^k_{i,j}) \in R^{m\times m},\ \mathcal{H}^k = \mathcal{H}^k(A) = (\sigma^k_{i,j}) \in R^{n\times n}, \qquad (1)$$

$$\mathcal{A}^k = \mathcal{A}^k(A) = (\alpha^k_{i,j}) \in R^{n\times m},\ \mathcal{A}dj^k = (\beta^{k-1}_{i,j}) \in R^{k\times k},$$

$\mathcal{A}dj^k$ — присоединенная матрица для блока $A[^{1\ldots k}_{1\ldots k}]$.

Известно основное минорное тождество [6].

Т е о р е м а 1. Основное минорное тождество.

Для матрицы $A \in R^{n\times m}$ и целых i, j, s, k, $0 \leqslant k < s \leqslant \min(n, m)$, $0 < i \leqslant n, 0 < j \leqslant m$, верно

$$\alpha^s \alpha^{k+1}_{ij} - \alpha^k a^{s+1}_{ij} = \sum_{p=k+1}^{s} \alpha^{k+1}_{ip}\delta^s_{pj}. \tag{2}$$

Т е о р е м а 2. *Для матрицы* $A \in R^{n\times m}$ и целых чисел s, k, $0 \leqslant k < s \leqslant \min(n, m)$, верно

$$\mathcal{U}[^{k+1\ldots s}_{k+1\ldots s}]\mathcal{G}^s[^{k+1\ldots s}_{s+1\ldots m}] = \alpha^s \mathcal{U}[^{k+1\ldots s}_{s+1\ldots m}]. \tag{3}$$

Д о к а з а т е л ь с т в о. Воспользуемся частным случаем тождества (2) при $i = k + 1$. Получим

$$\sum_{p=i}^{s} \alpha^i_{ip}\delta^s_{pj} = \alpha^s \alpha^i_{ij} - \alpha^{i-1}\alpha^{s+1}_{ij}.$$

Отметим, что по определению $a^{s+1}_{ij} = 0$ при $i \leqslant s$ и $\alpha^i_{ip} = 0$ при $p < i$. Поэтому $\forall i \leqslant s$ имеет место равенство

$$\sum_{p=k+1}^{s} \alpha^i_{ip}\delta^s_{pj} = \alpha^s \alpha^i_{ij}.$$

Т е о р е м а 3. *Для матрицы* $A \in R^{n\times m}$ и целых чисел s, k, $0 \leqslant k < s \leqslant \min(n, m)$, верно

$$\mathcal{H}^s[^{s+1\ldots n}_{k+1\ldots s}]\mathcal{L}[^{k+1\ldots s}_{k+1\ldots s}] = \alpha^s \mathcal{L}[^{s+1\ldots n}_{k+1\ldots s}]. \tag{4}$$

Д о к а з а т е л ь с т в о. По определению матриц миноров имеют место равенства

$$\mathcal{U}(A) = (\mathcal{L}(A^T))^T,\ \mathcal{L}(A) = (\mathcal{U}(A^T))^T,\ \mathcal{H}^s(A) = (\mathcal{G}^s(A^T))^T,\ \mathcal{G}^s(A) = (\mathcal{H}^s(A^T))^T$$

Применяя эти равенства и теорему 2, получим:

$$(\mathcal{H}^s[^{s+1\ldots n}_{k+1\ldots s}](A)\mathcal{L}[^{k+1\ldots s}_{k+1\ldots s}](A))^T = (\mathcal{L}[^{k+1\ldots s}_{k+1\ldots s}](A))^T(\mathcal{H}^s[^{s+1\ldots n}_{k+1\ldots s}](A))^T = \mathcal{U}[^{k+1\ldots s}_{k+1\ldots s}](A^T)\mathcal{G}^s[^{k+1\ldots s}_{s+1\ldots n}](A^T) =$$
$$\alpha^s \mathcal{U}[^{k+1\ldots s}_{s+1\ldots n}](A^T).$$

Отсюда следует

$$\mathcal{H}^s[^{s+1\ldots n}_{k+1\ldots s}](A)\mathcal{L}[^{k+1\ldots s}_{k+1\ldots s}](A) = (\alpha^s \mathcal{U}[^{k+1\ldots s}_{s+1\ldots n}](A^T))^T = \alpha^s \mathcal{L}[^{s+1\ldots n}_{k+1\ldots s}](A).$$

С л е д с т в и е 1. *Справедливы тождества для матриц миноров, которые выражают верхний правый блок* $\mathcal{U}$ *и нижний левый блок* $\mathcal{L}$*:*

$$\mathcal{U}[^{1\ldots k}_{1\ldots k}]\mathcal{G}^k[^{1\ldots k}_{k+1\ldots m}] = \alpha^k \mathcal{U}[^{1\ldots k}_{k+1\ldots m}],\quad \mathcal{H}^k[^{k+1\ldots n}_{1\ldots k}]\mathcal{L}[^{1\ldots k}_{1\ldots k}] = \alpha^k \mathcal{L}[^{k+1\ldots n}_{1\ldots k}]. \tag{5}$$

Эти тождества получаются из утверждений теорем 2 и 3, если положить $k = 0$, а затем переобозначить переменную s символом k.

3 Вычисление блоков матриц $\mathcal{L}$ и $\mathcal{U}$

Тождества (3) и (4) для $\mathcal{L}$ и $\mathcal{U}$ матриц, которые получены во втором параграфе, позволяют сформулировать рекурсивный алгоритм вычисления этих матриц. Для этого нужно воспользоваться блочным алгоритмом [4] вычисления присоединенной матрицы, в котором вычисляются все необходимые блоки присоединенной матрицы.

Пусть дана матрица $\mathcal{A}$ размера $2k \times 2k$, где k является некоторой степенью числа 2. Предположим, что все ведущие угловые миноры матрицы $\mathcal{A}$ отличны от нуля и пусть $\mathcal{A} = \begin{pmatrix} A & C \\ B & D \end{pmatrix}$ блочная запись матрицы $\mathcal{A}$. Требуется найти матрицы $\mathcal{U} = \begin{pmatrix} \mathcal{U}^{[1\ldots k]}_{[1\ldots k]} & \mathcal{U}^{[1\ldots k}_{[k+1\ldots 2k]} \\ 0 & \mathcal{U}^{[k+1\ldots 2k]}_{[k+1\ldots 2k]} \end{pmatrix}$, $\mathcal{L} = \begin{pmatrix} \mathcal{L}^{[1\ldots k]}_{[1\ldots k]} & 0 \\ \mathcal{L}^{[k+1\ldots 2k]}_{[1\ldots k]} & \mathcal{L}^{[k+1\ldots 2k]}_{[k+1\ldots 2k]} \end{pmatrix}$.

На первом шаге рекурсивный алгоритм вычисляет присоединенную матрицу $\mathcal{A}dj(A)$, $\mathcal{U}(A)$ и $\mathcal{L}(A)$ — $\mathcal{U}$ и $\mathcal{L}$ матрицы для блока A и новый блок $D' = (a^{k+1}_{ij})$, $i, j = k, k+1, \ldots, 2k$.

Известные тождества [6] позволяют вычислить блоки $\mathcal{G}$ и $\mathcal{H}$ матриц:

$$\mathcal{A}dj(A)C = \mathcal{G}^k{}^{[1\ldots k}_{[k+1\ldots 2k]}(\mathcal{A}) \text{ и } B\mathcal{A}dj(A) = \mathcal{H}^k{}^{[k+1\ldots 2k]}_{[1\ldots k}(\mathcal{A}). \tag{6}$$

Теоремы 2 и 3 для частного случая $k = 0$ и $s = k$ в применении к матрице $\mathcal{A}$ дают равенства:

$$\begin{aligned} \mathcal{U}^{[1\ldots k]}_{[1\ldots k]}(\mathcal{A}) \cdot \mathcal{G}^k{}^{[1\ldots k}_{[k+1\ldots 2k]}(\mathcal{A}) &= \alpha^k \mathcal{U}^{[1\ldots k}_{[k+1\ldots 2k]}(\mathcal{A}), \\ \mathcal{H}^k{}^{[k+1\ldots 2k]}_{[1\ldots k}(\mathcal{A}) \cdot \mathcal{L}^{[1\ldots k]}_{[1\ldots k]}(\mathcal{A}) &= \alpha^k \mathcal{L}^{[k+1\ldots 2k]}_{[1\ldots k}(\mathcal{A}). \end{aligned} \tag{7}$$

Эти равенства позволяют вычислить по одному квадратному блоку матриц $\mathcal{U}$ и $\mathcal{L}$.

Оставшиеся треугольные блоки $\mathcal{U}^{[k+1\ldots 2k]}_{[k+1\ldots 2k]}$ и $\mathcal{L}^{[k+1\ldots 2k]}_{[k+1\ldots 2k]}$ матриц $\mathcal{U}$ и $\mathcal{L}$ вычисляются рекурсивно на основе блока D'.

По теоремам 2 и 3 для матрицы $A \in R^{2k\times 2k}$ и произвольного $1 < s \leqslant k$ верно

$$\mathcal{U}^{[k+1\ldots k+s]}_{[k+1\ldots k+s]}\mathcal{G}^s{}^{[k+1\ldots k+s}_{[k+s+1\ldots k+2s]} = \alpha^s \mathcal{U}^{[k+1\ldots k+s}_{[k+s+1\ldots k+2s]},\ \mathcal{H}^s{}^{[k+s+1\ldots k+2s]}_{[k+1\ldots k+s}\mathcal{L}^{[k+1\ldots s]}_{[k+1\ldots s]} = \alpha^s \mathcal{L}^{[k+s+1\ldots k+2s]}_{[k+1\ldots k+s}. \tag{8}$$

Поэтому можно последовательно вычислять блоки матриц $\mathcal{U}$ и $\mathcal{L}$ для $s = 2, 4, 8, \ldots$. Необходимые блоки присоединенной матрицы вычисляются в рекурсивном алгоритме.

4 Алгоритм для разреженных матриц

В случае разреженной матрицы, когда ведущие миноры могут быть равны нулю, алгоритм [4] применять нельзя, так как он остановится на первом же нулевом ведущем элементе. Поэтому воспользуемся рекурсивным алгоритмом [10], [11]. Этот алгоритм нужно дополнить вычислением блоков L, D, U и двух новых матриц перестановок E_r и E_c. В результате этого присоединенное отображение, введенное в [10], станет необходимым для формулировки настоящего алгоритма $\mathcal{LDU}$ отображением.

Отображение $\mathcal{LDU}: R^{n\times n} \times (R\backslash 0) \to (R^{n\times n})^7$

$$(A, S, E_r, E_c, L, D, U) = \mathcal{LDU}(M, d), \tag{9}$$

при $n = 2^k$ назовем $\mathcal{LDU}$-отображением пары (M, d), если $M \in R^{n\times n}$, A — это присоединенная матрица, а S — ступенчатая форма матрицы M и отображение определено рекурсивно следующим образом.

При $M = 0$:

$$\mathcal{LDU}(0, d) = (dI, 0, 0, 0, 0, 0, 0). \tag{10}$$

Нулем мы всюду обозначаем тождественную перестановку.

При $k = 0$ и $M = a \neq 0$: $\mathcal{LDU}(a, d) = (d, a, 0, 0, a, a, a)$.

При $k > 0$ и $M \neq 0$ разобьем матрицу M на четыре равных блока $M = (M_{ij})$, $i, j \in \{1, 2\}$. Пусть

$$(A_{11}, S_{11}, Er_{11}, Ec_{11}, L_{11}, D_{11}, U_{11}) = \mathcal{LDU}(M_{11}, d). \tag{11}$$

Выполним перестановки $M_{12} = Er_{11}M_{12}$, $M_{21} = M_{21}Ec_{11}$, обозначим

$$M^1_{12} = A_{11}M_{12}/d,\ M^1_{21} = M_{21}Y_{11}/d.$$

Пусть

$$(A_{21}, S_{21}, Er_{21}, Ec_{21}, L_{21}, D_{21}, U_{21}) = \mathcal{LDU}(M^1_{21}, D^0_{11}), \tag{12}$$

$$(A_{12}, S_{12}, Er_{12}, Ec_{12}, L_{12}, D_{12}, U_{12}) = \mathcal{LDU}(\bar{I}_{11}M^1_{12}, D^0_{11}). \tag{13}$$

Выполним перестановки $A_{11} = Er_{12}A_{11}Er_{12}$, $S_{11} = S_{11}Ec_{21}$, $M_{22} = Er_{21}M_{22}Ec_{12}$, $M_{21} = Er_{21}M_{21}$, $M^1_{12} = Er_{12}M^1_{12}Ec_{12}$, $U_{11} = U_{11}Ec_{21}$, а также $M^1_{12} = M^1_{12}Ec_{22}$, $M_{22} = Er_{22}M_{22}Ec_{22}$, $M_{12} = M_{12}Ec_{12}Ec_{22}$.

Обозначим

$$M^1_{22} = (M_{22} - M_{21}E^T_{11}M^1_{12})/d,$$

$$M^2_{22} = A_{21}M^1_{22}Y_{12}/(D^0_{11})^2, d_s = D^0_{21}D^0_{12}/D^0_{11}$$

Пусть

$$(A_{22}, S_{22}, Er_{22}, Ec_{22}, L_{22}, D_{22}, U_{22}) = \mathcal{LDU}(\bar{I}_{21}M^2_{22}, d_s). \tag{14}$$

Выполним перестановки $A_{21} = Er_{22}A_{21}Er_{22}$, $S_{12} = S_{12}Ec_{22}$, $M^1_{22} = Er_{22}M^1_{22}Ec_{22}$, $M^2_{22} = Er_{22}M^2_{22}Ec_{22}$, $M_{21} = Er_{22}M_{21}$.

Обозначим

$$M^2_{11} = -S_{11}Y_{21}/D^0_{11},$$

$$M^2_{12} = S_{12}D^0_{21} + (I_{11}M^1_{12}D^0_{11}D^0_{21} - S_{11}E^T_{21}A_{21}M^1_{22})Y_{12}/(D^0_{11})^2,$$

$$M^3_{12} = M^2_{12}Y_{12}/(d_sD^0_{11}),\ M^3_{22} = S_{22} + I_{21}M^2_{22}Y_{12}/d_s,$$

$$L = (\mathbf{I}D^0_{11} - I_{11}M^1_{12}E^T_{12})A_{12}A_{11}D^0_{22}/(D^0_{11})^2,$$

$$Q = (\mathbf{I}D^0_{12}D^0_{21} - I_{21}M^2_{22}E^T_{22}D^0_{11})A_{22}/d_s,$$

$$F = S_{11}E^T_{21}D^0_{22} + M^2_{12}E^T_{22}A_{22}/d_s,$$

$$G = A_{21}(M^1_{22}E^T_{12}A_{12}d + M_{21}E^T_{11}D^0_{12}D^0_{11})A_{11}/(D^0_{11})^2d,$$

$$A = \begin{pmatrix} (L + FG/(D^0_{11}D^0_{21})/D^0_{12} & -FA_{21}/(D^0_{11}D^0_{21}) \\ -QG/(D^0_{11}D^0_{21}D^0_{12}) & QA_{21}/(D^0_{11}D^0_{21}) \end{pmatrix},$$

$$S = \begin{pmatrix} M^2_{11}D^0_{22}/D^0_{21} & M^3_{12} \\ S_{21}D^0_{22}/D^0_{21} & M^3_{22} \end{pmatrix}, E = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}.$$

Найдем матрицы перестановок строк и столбцов, которые получаются в результате перестановки четырех блоков и вычислим присоединенную матрицу и ступенчатую форму

$$Ec = PermutOfCol(rank_{11}, rank_{12}, rank_{21}, rank_{22}),$$

$$Er = PermutOfRow(rank_{11}, rank_{12}, rank_{21}, rank_{22}),$$

$$A = ErAEr,\ S = ErSEc.$$

Здесь $rank_{ij}$ обозначает ранг матрицы M_{ij}. Введем обозначение для матриц перестановок. Пусть e_{pq} обозначает матрицу, у которой элемент $(p.q)$ равен 1, а остальные элементы нулевые. Матрицы перестановок $E_{(i,j,s)} = \sum_{t=1}^{s} e_{i+t,j+t}$ имеют s единичных элементов. Матрицы перестановок E_{ij} определены так: $E_{11} = E_{(0,0,rank_{11})}$, $E_{12} = E_{(rank_{11},0,rank_{21})}$, $E_{21} = E_{(0,rank_{11},rank_{21})}$, $E_{22} = E_{(rank_{21},rank_{12},rank_{22})}$. Использованы обозначения

$$I_{ij} = E_{ij}E_{ij}^T,\ J_{ij} = E_{ij}^T E_{ij},\ Y_{ij} = D_{ij}^0\mathbf{I} - E_{ij}^T S_{ij},\quad i,j \in \{1,2\}.$$

Найдем перестановки, которые производились со строками и столбцами исходной матрицы, для этого вычислим композиции перестановок:

$$Ec = PermutOfMatrix(Ec, Ec_{11}, Ec_{12}, Ec_{21}, Ec_{22});$$

$$Er = PermutOfMatrix(Er, Er_{11}, Er_{12}, Er_{21}, Er_{22}).$$

Осталось вычислить L, D и U матрицы. Диагональные элементы матрицы D образуются из последовательностей элементов четырех блочных диагональных матриц, которые берутся в порядке D_{11}, D_{21}, D_{12}, D_{22}.

Перестановки блоков L и U матриц приведены на рисунках 1 и 2. Слева указано расположение блоков до перестановки, а справа указано расположение блоков после перестановки.

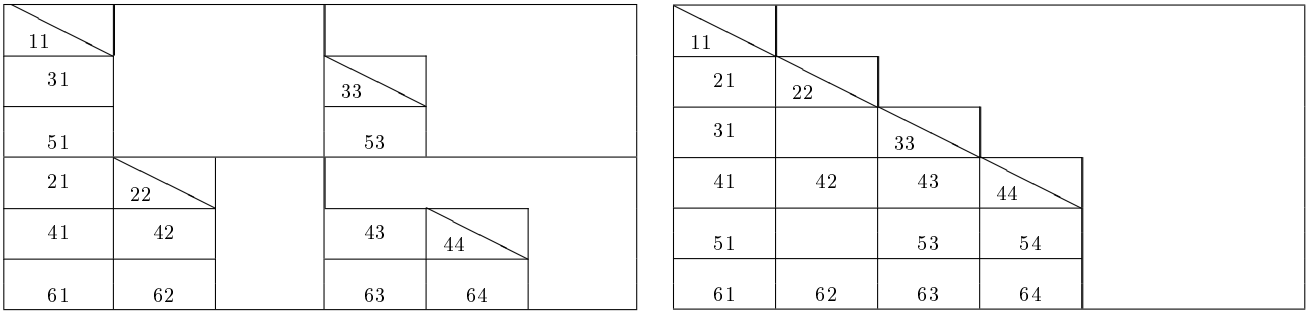

Рис. 1. Перестановки блоков матрицы L

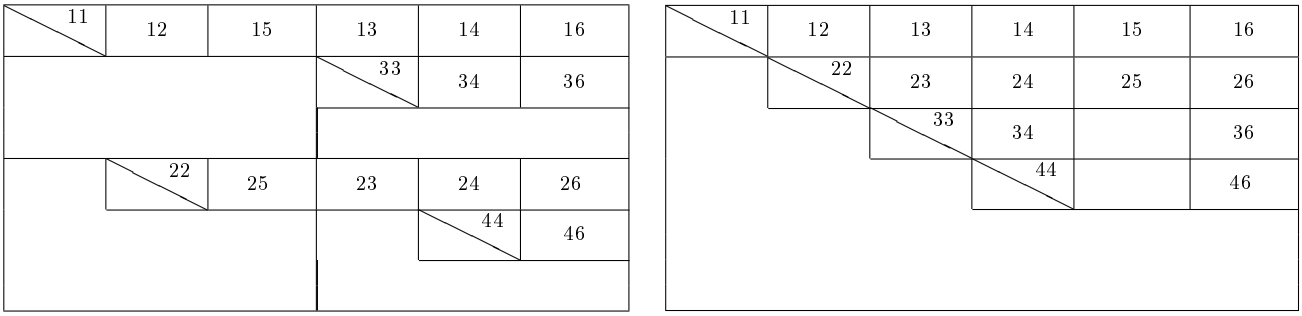

Рис. 2. Перестановки блоков матрицы U

Это перестановки блоков, которые производятся на каждом рекурсивном шаге. Если верхний левый блок имеет полный ранг, то выполнять перестановки блоков не требуется. После перестановок вычисляются новые блоки матриц L и U в соответствии с (7) и (8).

В результате получим

$$(A, S, Er, Ec, L, D, U) = \mathcal{LDU}(M, d).$$

Для произвольной матрицы $M \in R^{n\times n}$ LDU отображение

$$(A, S, Er, Ec, L, D, U) = \mathcal{LDU}(M, 1)$$

предъявляет присоединенную невырожденную матрицу A, ступенчатую матрицу S, матрицы перестановок E_r и E_c и матрицы L, D и U.

LDU разложение для матрицы M будет иметь вида

$$M = (E_r^T L E_r^T)(E_r^T D^{\#} E_c^T)(E_c^T U E_c^T),$$

где диагональная матрица $D^{\#}$ получена из элементов диагональной матрицы D следующим образом

$$D^{\#} = \mathbf{diag}(D_{11}, D_{11}D_{22}, D_{22}D_{33}, \ldots, D_{r-1,r-1}D_{rr})^{-1}.$$

Пусть задана матрица B и $M = I'B$, где I' введенная выше матрица перестановок. Если для матрицы M получено LDU разложение, то искомое обобщенное разложение Брюа VwU для матрицы B будет образовано матрицами

$$V = (I' E_r^T L E_r^T I'),\ w = (I' E_r^T D^{\#} E_c^T),\ U = (E_c^T U E_c^T).$$

5 Заключение

Алгоритм LDU разложения, который излагается в этой работе был положен в основу программы, которая используется в системе компьютерной алгебры Mathpar [13], [14] для вычисления LDU разложения и разложения Брюа.

Данный блочно-рекурсивный алгоритм предназначен для разработки на его основе параллельных программ, для таких матриц, которые по своим размерам недоступны для однопроцессорных машин. Планируется разработка таких параллельных программ в системе Mathpar.

ЛИТЕРАТУРА

1. *Grigoriev D.* Analogy of Bruhat decomposition for the closure of a cone of Chevalley group of a classical serie // Soviet Math. Dokl. Vol. 23. N. 2. 1981. P 393–397.

2. *Kolotilina L.Yu.* Sparsity of Bruhat decomposition factors of nonsingular matrices // Notes of Scientific Seminars of LOMI. V. 202. 1992. P. 5-17.

3. *Strassen V.* Gaussian Elimination is not optimal // Numerische Mathematik. **13**. 1969. P. 354–356.

4. *Малашонок Г.И.* Быстрый алгоритм вычисления присоединенной матрицы // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 5. Вып. 1. 2000. С. 142-146.

5. *Malaschonok G.I.* Effective Matrix Methods in Commutative Domains // Formal Power Series and Algebraic Combinatorics. Springer, Berlin. 2000. P. 506-517.

6.*Малашонок Г.И.* Матричные методы вычислений в коммутативных кольцах. Монография. Тамбов: Изд-во ТГУ им. Г.Р. Державина, 2002.

7. *Akritas A. and Malaschonok G.* Computation of Adjoint Matrix // Fourth International Workshop on Computer Algebra Systems and Applications (CASA 2006), LNCS 3992. Springer, Berlin. 2006. P. 486-489.

8. *Bunch J., Hopkroft J.* Triangular factorization and inversion by fast matrix multiplication. Mat. Comp. V. 28, 231-236 (1974)

9. *Malaschonok G.I.* Parallel Algorithms of Computer Algebra // Materials of the conference dedicated for the 75 years of the Mathematical and Physical Dep. of Tambov State University. (November 22-24, 2005). Tambov, TSU. 2005. P. 44-56.

10. *Малашонок Г.И.* О вычислении ядра оператора, действующего в модуле // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 13. Вып. 1. 2008. С. 129-131.

11. *Malaschonok G.I.* Fast matrix decomposition in parallel computer algebra // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 15. Вып. 4. 2010. С.1372-1385.

12. *Malaschonok G. I.* Fast Generalized Bruhat Decomposition // Computer Algebra in Scientific Computing, LNCS 6244, Springer, Berlin. 2010. P. 194-202.

13. *Malaschonok G.I.* Project of Parallel Computer Algebra // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 6. 2010. С. 1724-1729.

14. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 1. 2010. С. 322-327.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

ON FAST GENERALIZED BRUHAT DECOMPOSITION IN THE DOMAINS

© **Malaschonok Gennadi Ivanovich**
Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Doctor of Physics and Mathematics, Professor of Mathematical Analysis Department, e-mail: malaschonok@gmail.com

Key words: generalized Bruhat decomposition, block recursive algorithm, commutative domainconstructive mathematics, conception of short-range interaction, parallel algorithms, cluster computations, Mathpar algorithms library.

We consider a deterministic recursive algorithm for computing the generalized Bruhat decomposition for matrices over commutative domains. The computational complexity of this algorithm is equals the complexity of matrix multiplication. The algorithm is included in the Mathpar algorithms library.

УДК 519.688

О НАХОЖДЕНИИ ОБЩЕГО И ЧАСТНОГО РЕШЕНИЙ НЕОДНОРОДНОЙ СИСТЕМЫ ОБЫКНОВЕННЫХ ЛИНЕЙНЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ С ПОСТОЯННЫМИ КОЭФФИЦИЕНТАМИ

© **М. А. Рыбаков**

Ключевые слова: неоднородная система обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами, аналитический метод решения, преобразование Лапласа, система Mathpar.

Обсуждается алгоритм получения символьного аналитического решения неоднородной системы обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами. Разработанный алгоритм позволяет получить как частное, так и общее решение системы дифференциальных уравнений. Решение системы находится в аналитическом виде и может быть найдено с требуемой точностью. Полученный алгоритм эффективен для решения систем дифференциальных уравнений большого размера. Алгоритм входит в состав библиотеки алгоритмов системы Mathpar. Приводятся примеры решения систем дифференциальных уравнений в системе Mathpar.

1 Введение

Одной из актуальных задач компьютерной алгебры является задача решения систем линейных дифференциальных уравнений с постоянными коэффициентами. Аналитический метод решения таких систем уравнений является сложной задачей, поэтому обычно прибегают к приближенным численным методам. Сегодня масштаб задач изменился и требует других подходов. Одним из современных аппаратов, применяемых для решения дифференциальных уравнений и их систем, является преобразование Лапласа.
Преобразование Лапласа позволяет переходить от решения системы дифференциальных уравнений к решению системы алгебраических уравнений и в итоге получать решение в символьном виде.

В параграфе 2 сформулирована постановка задачи решения неоднородных систем обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами в общем виде.

В параграфе 3 приводится алгоритм решения неоднородных систем обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами с помощью преобразования Лапласа. Этот алгоритм состоит из трёх этапов: прямое преобразование Лапласа, решение полученной системы алгебраических уравнений и обратное преобразование Лапласа. Эти этапы подробно рассмотрены в параграфах 4, 5, 6.

В параграфе 7 описана программная реализация полученного алгоритма в системе компьютерной алгебры Mathpar. Приведены названия соответствующих классов и методов, реализованных в этой системе. Приводятся примеры нахождения общего и частного решений неоднородных систем обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами с помощью разработанных программ.

В параграфе 8 приводится описание существенных особенностей системы Mathpar, которые позволяют решать системы обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами. Приведены примеры решения систем линейных дифференциальных уравнений в системе Mathpar.

2 Постановка задачи

Пусть задана неоднородная система обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами:

$$\sum_{j=1}^{n} D_{ji}(t)x_i(t) = f_i(t), D_{ji}(t) = \sum_{k=0}^{m} a_{kji}\frac{d^k}{dt^k}, i = 1, ..., m, a_{kji} \in \mathbb{R}, n, m \in \mathbb{N}, \quad (1)$$

где a_{kji} — действительные числа, $f_i(t), x_i(t)$ — ограниченные на $\mathbb{R}_+$ функции, имеющие конечное число точек разрыва I рода и удовлетворяющие условиям: $f_i(t) \equiv 0$ при $t < 0$, $|f_i(t)| < Me^{s_0 t}$ при $t > 0$, где $M > 0$, $s_0 \geqslant 0$ — некоторые действительные постоянные.

Систему (1) можно записать в матричном виде:

$$A(t)X(t) = F(t), \quad (2)$$

где

$$A(t) = \begin{pmatrix} D_{11}(t) & D_{12}(t) & \dots & D_{1m}(t) \\ D_{21}(t) & D_{22}(t) & \dots & D_{2m}(t) \\ \dots & \dots & \dots & \dots \\ D_{n1}(t) & D_{n2}(t) & \dots & D_{nm}(t) \end{pmatrix},$$

$$X(t) = [x_1(t), ..., x_n(t)]^T, \ F(t) = [f_1(t), ..., f_n(t)]^T.$$

Пусть $x_i^{(k)}(t)$ обозначает k-тую производную функции $x_i(t)$, а числа x_{0i}^k определяют начальные условия:

$$x_i^{(k)}(0) = x_{0i}^k, \quad (3)$$

где $k = 1, 2, ..., m-1, \ i = 1, 2, ..., m, \ x_{0i}^k \in \mathbb{R}$.

Будем полагать, что в общем случае каждая из функций $f_i(t)$ в правой части может иметь вид конечной суммы:

$$f_i(t) = \sum_j p_{ij}(t)e^{\delta_{ij}t}\sin^{\mu_{ij}}(\gamma_{ij}t)\cos^{\nu_{ij}}(\beta_{ij}t)UnitStep(t - \alpha_{ij}),$$

где $\alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij} \in \mathbb{R}, \mu_{ij}, \nu_{ij} \in \mathbb{N}, p_{ij}(t)$ — полином переменной t, функция $UnitStep(t)$ принимает значение 1 для неотрицательных аргументов и значение 0 для остальных.

Требуется найти решение X(t) системы (1) или (2), удовлетворяющее условиям (5), в аналитическом виде.

Решение системы (1) называется частным, когда начальные условия заданы числами. Решение системы (1) называется общим решением, если начальные условия входят в решение свободными переменными, которые можно подобрать так, чтобы решение $X(t)$ удовлетворяло произвольным начальным условиям.

3 Алгоритм Лапласа

Алгоритм состоит из трёх этапов [1-9].

Этап I. Прямое преобразование Лапласа системы дифференциальных уравнений.

Преобразования Лапласа для функции $f(t)$:

$$\mathcal{F}(p) = \int_0^\infty f(t)e^{-pt}dt, \quad p \in \mathbb{C}. \tag{4}$$

В результате преобразования функций, стоящих в левой и правой частях системы дифференциальных уравнений (1), и начальных условий (5) по формуле (4) получаем систему алгебраических уравнений:

$$\mathcal{A}(p)\mathcal{X}(p) - \mathcal{B}(p) = \mathcal{F}(p). \tag{5}$$

Здесь $\mathcal{A}(p)$ — образ левой части системы (1), $\mathcal{F}(p)$ — образ правой части системы (1), $\mathcal{B}(p)$ — вектор, появляющийся при введении начальных условий (5).

Этап II. Решение полученной алгебраической системы.

Решение алгебраической системы (5) ищем в виде

$$\mathcal{X}(p) = \mathcal{A}(p)^{-1}(\mathcal{F}(p) + \mathcal{B}(p)). \tag{6}$$

Результатом является вектор дробно-рациональных функций от p.

Этап III. Обратное преобразование Лапласа.

Обратное преобразование Лапласа для функции $\mathcal{F}(p)$:

$$f(t) = L^{-1}\{\mathcal{F}\} = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} e^{pt}\mathcal{F}(p)dp.$$

Искомое решение системы (1) задается вектором $X(t)$:

$$X(t) = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} e^{pt}\mathcal{X}(p)dp. \tag{7}$$

В следующих параграфах подробно рассматривается каждый этап.

4 Прямое преобразование Лапласа

Пусть $\mathcal{F}(p)$ — образ функции $f(t)$ при прямом преобразовании Лапласа. В результате прямого преобразования Лапласа производной n-ого порядка функции $f(t)$ получим полином степени n

$$\int_0^\infty f^{(n)}(t)e^{-pt}dt = p^n\mathcal{F}(p) - p^{n-1}f(0) - p^{n-2}f'(0) - \ldots - f^{(n-1)}(0). \tag{8}$$

В результате прямого преобразования Лапласа левая часть системы дифференциальных уравнений преобразуется в матрицу полиномов $\mathcal{A}(p)$ одной действительной переменной p.

$$\int_0^\infty \sum_{j=1}^{n}\sum_{k=0}^{m} a_{kji}\frac{d^k}{dt^k}x_i(t)e^{-pt}dt = \sum_{j=1}^{n}\mathcal{D}_{ji}(p)x_i(p),$$

где $\mathcal{D}_{ji}(p) = \sum_{i=1}^{m} a_{kji}p^k$.

$$\mathcal{A}(p) = \begin{pmatrix} \mathcal{D}_{11}(p) & \mathcal{D}_{12}(p) & \ldots & \mathcal{D}_{1m}(p) \\ \mathcal{D}_{21}(p) & \mathcal{D}_{22}(p) & \ldots & \mathcal{D}_{2m}(p) \\ \ldots & \ldots & \ldots & \ldots \\ \mathcal{D}_{n1}(p) & \mathcal{D}_{n2}(p) & \ldots & \mathcal{D}_{nm}(p) \end{pmatrix}. \tag{9}$$

Результатом прямого преобразования Лапласа (4) правой части будет вектор функций.

$$\mathcal{F}(p) = [f_1(p), \ldots, f_n(p)]^T,$$

Каждая из функций $f_i(p)$ является суммой произведений экспоненциальных и дробно-рациональных функций

$$f_i(p) = \sum_{j=1}^{n}\frac{Q_{ij}(p)e^{\alpha_{ij}p}}{L(p)}, \text{ где } Q_{ij}(p), L(p) \text{ — полиномы.} \tag{10}$$

Результатом преобразования начальных условий будет сумма произведений полиномов от p на свободные переменные, обозначающие начальные условия (5):

$$\mathcal{B}(p) = \sum_{i=1}^{n}\sum_{k=0}^{m-1} d_{ki}(p)x_{0i}^k, \quad d_{ik}(p) = \sum_{i=k}^{m-1} a_{i+1,i}p^{i-k}. \tag{11}$$

В результате преобразования Лапласа системы обыкновенных линейных дифференциальных уравнений получаем алгебраическую систему линейных уравнений

$$\mathcal{X}(p) = \mathcal{A}(p)^{-1}(\mathcal{F}(p) + \mathcal{B}(p)).$$

5 Решение алгебраической системы линейных уравнений

Для матрицы полиномов $\mathcal{A}(p) \in \mathbb{C}[p]^{n\times m}$, заданной формулой (9), вычисляем обратную матрицу $\mathcal{A}^{-1} = \mathcal{A}^*(p)/det(\mathcal{A}(p))$, где $\mathcal{A}^*(p)$ — присоединённая матрица, $det(\mathcal{A}(p))$ — определитель.

Полином $det(\mathcal{A}(p))$ раскладываем на линейные сомножители в области $\mathbb{C}$:

$$det(\mathcal{A}(p)) = \prod_{k=0}^{r}(p-p_k)^{\varepsilon_k},\ \text{где}\ r \leqslant nm,\ p_k \in \mathbb{C},\ \varepsilon_k \in \mathbb{N}. \tag{12}$$

Раскладываем дробь $1/det(\mathcal{A}(p))$ в сумму простых дробей в области $\mathbb{C}$:

$$\frac{1}{det(\mathcal{A}(p))} = \sum_{k=0}^{r}\frac{\zeta_k}{(p-p_k)^{\varepsilon_k}},\ \text{где}\ \zeta_k \in \mathbb{C}. \tag{13}$$

Обозначим через $H(p) = \mathcal{F}(p)+\mathcal{B}(p)$ и $H(p) = [h_1,\ldots,h_n]^T$. Тогда, используя формулы (10) и (11), получим:

$$h_i = \sum_{j=1}^{n}\frac{Q_{ij}(p)e^{\alpha_{ij}p}}{L(p)} + \sum_{i=1}^{n}\sum_{k=0}^{m-1} d_{ki}(p)x_{0i}^k, d_{ik}(p) = \sum_{i=k}^{m-1} a_{i+1,i}p^{i-k}.$$

Пусть $V = [v_1,\ldots,v_n]^T$ и $\mathrm{V} = \mathcal{A}^*(p)\cdot H$. Тогда:

$$v_i = \sum_{j=1}^{n}\frac{U_{ij}(p)e^{\alpha_{ij}p}}{O(p)},\ \text{где}\ U_{ij}(p),\ O(p) — \text{полиномы}.$$

Разложим v_i в сумму простых дробей в области $\mathbb{C}$:

$$v_i = \sum_{k=0}^{r}\frac{\xi_k e^{\alpha_{ij}p}}{(p-p_k)^{\tau_k}},\ \text{где}\ \xi_k \in \mathbb{C},\ \tau_k \in \mathbb{N}. \tag{14}$$

Найдем решение $\mathcal{X}(p) = V\cdot 1/det(\mathcal{A}(p))$,

$$\mathcal{X}(p) = V\cdot\sum_{k=0}^{r}\frac{\zeta_k}{(p-p_k)^{\varepsilon_k}}.$$

Пусть $\mathcal{X}(p) = [\chi_1,\ldots,\chi_n]^T$. Тогда

$$\chi_i(p) = \sum_{k=0}^{r}\frac{\eta_k e^{\alpha_{ij}p}}{(p-p_k)^{\varepsilon_k}},\ \text{где}\eta_k \in \mathbb{C}. \tag{15}$$

З а м е ч а н и е 1. Разложение (12) на множители определителя $det(\mathcal{A}(p))$ требует вычисления корней многочлена. Корни многочлена находятся приближённо. Точность вычислений корней влияет на погрешность решения $X(t)$ системы (1). Зависимость погрешности получаемого решения системы (1) от точности вычисления корней исследована в работе [10].

6 Обратное преобразование Лапласа

Для каждого элемента вектора $\mathcal{X}(p) = [\chi_1(p),\ldots,\chi_n(p)]^T$ находим обратное преобразование Лапласа (7) $X(t) = [x_1(t),\ldots,x_n(t)]^T$:

$$x_i(t) = L^{-1}\{\chi_i(p)\},\ \text{где}\ i = 1,2,\ldots,n.$$

7 Пакет программ и примеры решения систем дифференциальных уравнений

7.1 Пакет программ

Рассмотренный в параграфах 1 — 6 алгоритм был программно реализован в системе компьютерной алгебры Mathpar [11-14].

Разработанный программный пакет *laplaceTransform* состоит из трёх основных классов.

Класс SystemLDE включает в себя процедуры, предназначенные для решения систем дифференциальных уравнений [15-16]. Процедура `systDifEquationToMatrix` вычисляет полиномиальную матрицу $\mathcal{A}(p)$ для системы (1), которая получается в результате преобразования Лапласа (4) левой части системы (1). Процедура `initCondLaplaceTransform` вычисляет вектор-функцию $\mathcal{B}(p)$, которая получается в результате преобразования Лапласа начальных условий. Процедура `primeFractionForSingle` предназначена для разложения дроби $1/f(p)$ с факторизованным знаменателем $f(p)$ в сумму простых дробей. Числители этих дробей находятся методом неопределённых коэффициентов. Процедура `primeFractions` предназначена для разложения суммы дробей $\sum_{i=1}^{n} 1/f_i(p)$, у которых факторизованы знаменатели $f_i(p)$, в сумму простых дробей.

Класс LaplaceTransform включает в себя процедуры, предназначенные для вычисления прямого преобразования Лапласа функций. Процедура `vectorLaplaceTransform` вычисляет прямое преобразования Лапласа (4) правой части системы (1).

Класс InverseLaplaceTransform включает в себя процедуры, предназначенные для вычисления обратного преобразования Лапласа. Процедура `vectorInverseLaplaceTransform` вычисляет обратное преобразования Лапласа для вектор-функции $\mathcal{X}(p)$. В результате получаем решение $X(t)$ системы (1).

Программный пакет *laplaceTransform* использует следующие программные пакеты системы Mathpar: *number*, *matrix*, *polynom*, *func*.

Пакет *number* предназначен для задания числовых множеств и операций над ними. Кроме того, в пакете *number* имеются два специализированных класса — Product и SumOfProducts. Класс Product включает в себя процедуры, предназначенные для операций над объектами вида:

$$\prod_i \alpha_i^{n_i}, \tag{16}$$

где $n, i \in \mathbb{N}, \alpha_i^{n_i}$ — скалярный объект из системы Mathpar (число, полином, функция). Все элементы в произведении (16) упорядочены некоторым образом. Класс SumOfProducts включает в себя процедуры, предназначенные для операций над объектами вида:

$$\sum_j \prod_i \alpha_{ij}^{n_{ij}}. \tag{17}$$

Процедура `primeFractions` класса SumOfProducts производит разложение объектов класса Product в сумму простых дробей. Процедура `jointExpPolSort` упрощает и сортирует в определённом порядке объекты класса Product, которые являются слагаемыми в объекте класса SumOfProducts.

Пакет *matrix* предназначен для операций над числовыми и функциональными матрицами. Процедура `matrixAdjointDet` пакета *matrix* позволяет вычислить определитель и

присоединённую матрицу для матрицы полиномов (9). Процедура `solveLAE` вычисляет решение системы (5).

Пакет *polynom* предназначен для операций над полиномами. Процедура `factorOfPol` раскладывает полином одной переменной на множители в поле комплексных чисел.

Пакет *func* предназначен для операций над композициями трансцендентных и рациональных функций.

7.2 Пример нахождения частного решения системы LDE

Дана система уравнений:

$$\begin{cases} x''(t) + y'(t) = sh(t) - sin(t) - t, \\ y''(t) + x'(t) = ch(t) - cos(t). \end{cases} \tag{18}$$

Даны начальные условия:

$$x(0) = 0;\ x'(0) = 2;\ y(0) = 1;\ y'(0) = 0. \tag{19}$$

Покажем отдельные этапы решения и используемые процедуры.

1. Прямое преобразование Лапласа.

После применения прямого преобразования Лапласа (4) к левой части системы (18) и при использовании процедуры `systDifEquationToMatrix` получим полиномиальную матрицу:

$$A(p) = \begin{pmatrix} p^2 & p \\ p & p^2 \end{pmatrix}.$$

В результате применения процедуры `vectorLaplaceTransform`, который вычисляет прямое преобразования Лапласа (4) правой части системы (18), получим:

$$\mathcal{F}(p) = [\frac{1}{p^2-1} - \frac{1}{p^2+1} - \frac{1}{p^2}, \frac{p}{p^2-1} - \frac{p}{p^2+1}]^T. \tag{20}$$

Процедура `initCondLaplaceTransform` вычисляет прямое преобразование Лапласа начальных условий системы (18):

$$\mathcal{B}(p) = [3, p]^T. \tag{21}$$

Суммируя векторы (20) и (21), находим образ правой части системы (18) в виде суммы дробей:

$$H = \left[\frac{1}{p^2-1} + \frac{-1}{p^2+1} + \frac{-1}{p^2} + 3,\ \frac{p}{p^2-1} + \frac{-p}{p^2+1} + p\right]^T. \tag{22}$$

2. Решение алгебраической системы.

Используя процедуру `matrixAdjointDet`, вычисляем определитель и присоединённую матрицу для матрицы полиномов $A(p)$:

$$A^*(p) = \begin{pmatrix} p^2 & -p \\ -p & p^2 \end{pmatrix},$$

$$det(A(p)) = p^4 - p^2.$$

Применяя процедуру `factorOfPol`, раскладываем $det(A(p))$ на множители в поле комплексных чисел:

$$det(A(p)) = p^2(p-1)(p+1).$$

С помощью процедуры `primeFractionForSingle` раскладываем дробь $1/det(A(p))$ с факторизованным знаменателем $det(A(p))$ в сумму простых дробей. Числители этих дробей находятся методом неопределённых коэффициентов. Для разложения дроби $1/det(A(p))$ нужно решить систему:

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix} \cdot Y = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{23}$$

Для решения системы (23) применяется процедура `solveLAE`.

В результате получим вектор решения: $Y = [0, -1, 0.5, 0.5]^T$ и искомое разложение в сумму простых дробей:

$$\frac{1}{det(A(p))} = \frac{-1}{p^2} + \frac{-0.5}{p+1} + \frac{0.5}{p-1}.$$

Получаем решение алгебраической системы:

$$\mathcal{X}(p) = A^*(p) \cdot (1/det(A(p))) \cdot H,$$

где H задается формулой (22).

В дальнейших вычислениях мы избегаем операций в поле дробно-рациональных функций и все вычисления производим над полиномами.

Разложим элементы вектора H, заданного формулой (22), в сумму простых дробей, применяя процедуру `primeFractions`. Получим вектор элементы которого являются объектами класса SumOfProducts (17):

$$H = [1 \cdot (p^2-1)^{-1} + (-1) \cdot (p^2+1)^{-1} + (-1) \cdot (p^{-2}) + 3, p \cdot (p^2-1)^{-1} + (-p) \cdot (p^2+1)^{-1} + p]^T.$$

Домножим вектор H на скаляр $1/det(A(p))$, применим процедуры `primeFractions` и `jointExpPolSort`. В результате получим вектор V, у которого элементы являются упорядоченными суммами простых дробей.

$$V = [p^{-4} + 0.75(p-1)^{-1} + (-0.75)(p+1)^{-1} + (-1)(p^2-1)^{-1} + (-0.5)(p^2+1)^{-1},$$

$$p^{-1} + 0.25(p-1)^{-1} + (-0.5p - 0.25)(p+1)^{-1} + (-p)(p^2-1)^{-1} + (-0.5p)(p^2+1)^{-1}]^T.$$

Найдем решение $\mathcal{X}(p) = V \cdot A^*(p)$. К каждому компоненту вектора решения $\mathcal{X}(p)$ применим процедуру `jointExpPolSort`:

$$\mathcal{X}(p) = \begin{pmatrix} \frac{1}{p^2} + \frac{0.5}{p-1} + \frac{-0.5}{p+1} \\ \frac{-1}{p^3} + \frac{0.5}{p-i} + \frac{0.5}{p+i} \end{pmatrix}.$$

3. Обратное преобразование Лапласа.

Для вычисления обратного преобразования Лапласа для каждого элемента вектора $\mathcal{X}(p)$ применим процедуру `vectorInverseLaplaceTransform`. Тогда получим решение $X(t) = [x(t), y(t)]$ системы (18):

$$\begin{cases} x(t) &= t + 0.5e^{t} - 0.5e^{-t}, \\ y(t) &= (-t^2/2) + 0.5e^{it} + 0.5e^{-it}. \end{cases}$$

7.3 Пример нахождения общего решения системы LDE

Дана система уравнений:

$$\begin{cases} x'(t) = y(t) - z(t), \\ y'(t) = x(t) + y(t), \\ z'(t) = x(t) + z(t). \end{cases} \tag{24}$$

Даны начальные условия:

$$x(0) = a;\ y(0) = b;\ z(0) = c. \tag{25}$$

Покажем отдельные этапы решения и используемые процедуры.

1. Прямое преобразование Лапласа.

Применяем преобразование Лапласа (4) к левой части системы (24), для этого используем процедуру `systDifEquationToMatrix`:

$$A(p) = \begin{pmatrix} 1 & p & -1 \\ 0 & -1 & p-1 \\ p-1 & -1 & 0 \end{pmatrix}.$$

Применяем преобразование Лапласа (4) к правой части системы (24), для этого используем процедуру `vectorLaplaceTransform`:

$$\mathcal{F}(p) = [0/p, 0/p, 0/p]^T. \tag{26}$$

Процедура `initCondLaplaceTransform` вычисляет прямое преобразование Лапласа начальных условий системы (24):

$$\mathcal{B}(p) = [a, b, c]^T. \tag{27}$$

Суммируя векторы (26) и (27), находим образ правой части системы (24) в виде суммы дробей:

$$H = \mathcal{F}(p) + \mathcal{B}(p) = [0/p + a, 0/p + b, 0/p + c]^T.$$

2. Решение алгебраической системы.

Используя процедуру `matrixAdjointDet`, вычисляем определитель и присоединённую матрицу для матрицы полиномов $A(p)$:

$$A^*(p) = \begin{pmatrix} p-1 & 1 & p^2-p-1 \\ p^2-2p+1 & p-1 & -p+1 \\ p-1 & p^2-p+1 & -1 \end{pmatrix},$$

$$det(A(p)) = p^3 - 2p^2 + p.$$

Применяя процедуру `factorOfPol`, раскладываем $det(A(p))$ на множители в поле комплексных чисел:

$$det(A(p)) = p(p-1)^2.$$

С помощью процедуры `primeFractionForSingle` раскладываем дробь $1/det(A(p))$ с факторизованным знаменателем $det(A(p))$ в сумму простых дробей. Числители этих дробей находятся методом неопределённых коэффициентов. Для разложения дроби $1/det(A(p))$ нужно решить систему:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ -2 & -1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & -2 & -1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot Y = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{28}$$

Для решения системы (28) применяется процедура `solveLAE`.

В результате получим вектор решения: $Y = [1, -1, 1]^T$ и искомое разложение в сумму простых дробей:

$$\frac{1}{p^3 - 2p^2 + p} = \frac{1}{p} + \frac{(-1)}{p-1} + \frac{1}{(p-1)^2}.$$

Получаем решение алгебраической системы:

$$\mathcal{X}(p) = (1/det(A(p))) \cdot H \cdot A^*(p).$$

В дальнейших вычислениях мы избегаем операций в поле дробно-рациональных функций и все вычисления производим над полиномами.

Разложим элементы вектора H, заданного формулой (22), в сумму простых дробей, применяя процедуру `primeFractions`. Получим вектор, элементы которого являются объектами класса SumOfProducts (17):

$$H = [a, b, c]^T.$$

Домножим вектор H на скаляр $1/det(A(p))$, применим процедуры `primeFractions` и `jointExpPolSort`. В результате получим вектор V, у которого элементы являются упорядоченными суммами простых дробей.

$$V = [ap^{-1} + (-a)(p-1)^{-1} + a(p-1)^{-2}, bp^{-1} + (-b)(p-1)^{-1} + b(p-1)^{-2},$$

$$cp^{-1} + (-c)(p-1)^{-1} + c(p-1)^{-2}]^T.$$

Найдем решение $\mathcal{X}(p) = V \cdot A^*(p)$. К каждому компоненту вектора решения $\mathcal{X}(p)$ применим процедуру `jointExpPolSort`:

$$\mathcal{X}(p) = \begin{pmatrix} \frac{(b+(-a)+(-c))}{p} + \frac{((-b)+a+2c)}{p-1} + \frac{(b+(-c))}{(p-1)^2} \\ \frac{((-b)+c+a)}{p} + \frac{(b+(-c))}{p-1} \\ \frac{((-c)+(-a)+b)}{p} + \frac{(c+a)}{p-1} + \frac{((-c)+b)}{(p-1)^2} \end{pmatrix}.$$

3. Обратное преобразование Лапласа.

Для вычисления обратного преобразования Лапласа для каждого элемента вектора $\mathcal{X}(p)$ применим процедуру `vectorInverseLaplaceTransform`. Тогда получим решение $X(t) = [x(t), y(t), z(t)]$ системы (24):

$$\begin{cases} x(t) &= ((-b) + c + a) + (b + (-c))e^t, \\ y(t) &= ((-c) + (-a) + b) + (c + a)e^t + ((-c) + b)e^t t, \\ z(t) &= (b + (-a) + (-c)) + ((-b) + a + 2c)e^t + (b + (-c))e^t t. \end{cases}$$

Это общее решение позволяет легко получить любое частное решение. Например, получим частное решение системы дифференциальных уравнений (24) при начальных условиях $x(0) = 1,\ y(0) = 2,\ z(0) = 3$. Для этого подставим $a = 1,\ b = 2,\ c = 3$ в общее решение:

$$\begin{cases} x(t) &= 2 - e^t, \\ y(t) &= -2 + 4e^t - te^t, \\ z(t) &= -2 + 5e^t - te^t. \end{cases}$$

8 Применение системы Mathpar

Разработанный программный пакет входит в систему компьютерной алгебры Mathpar [11]. Веб-система Mathpar предназначена для проведения символьных вычислений в инженерных расчётах, научных исследованиях и образовании. Система Mathpar позволяет оперировать с функциями и функциональными матрицами, получать как точные численно-аналитические решения, так и решения, у которых числовые коэффициенты в аналитических выражениях имеют требуемую точность.

Для решения неоднородной системы обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами в системе Mathpar необходимо:
1) задать пространство переменных ($SPACE$);
2) задать систему уравнений ($systLDE$);
3) задать начальные условия ($initCond$);
4) получить решение ($solveLDE$).

Пространство переменных определяется числовым множеством и именами переменных. Пользователь может сменить окружение, задав новое алгебраическое пространство переменных с помощью команды установки, например, « $SPACE = R64[x, y, z]$ » или « $SPACE = R64[x, y, z]$ ». Существует возможность установить разные числовые множества. Числовое множество $R64$ — это 8-байтовые числа с плавающей точкой. Числовое множество R — это множество приближенных действительных чисел с плавающей точкой, у которых число точных десятичных позиций после запятой, состоит из определяемого произвольного числа десятичных цифр, количество которых может задавать пользователь.

Кроме того, можно использовать вспомогательные средства системы Mathpar, такие как:
1) если, установлено числовое множество R, то команда « $ACCURACY = m$ » устанавливает число точных десятичных позиций после запятой (m);
2) команда настройки окружения « $FLOATPOS = n$ » устанавливает число десятичных знаков после запятой (n), которые должны появиться при выводе числового результата приближенных вычислений. Этот параметр задает только количество знаков при выводе ответа, но не влияет на точность вычислений.

Пример 1. Решить систему уравнений (18) с начальными условиями (19).

Ввод данных для решения задачи в системе Mathpar

$SPACE = R64[t];$
$g = systLDE(d(x,t,2) + d(y,t) = sh(t) - sin(t) - t, d(y,t,2) + d(x,t) = ch(t) - cos(t));$
$f = initCond(d(x,t,0,0) = 0, d(x,t,0,1) = 2, d(y,t,0,0) = 1), d(y,t,0,1) = 0);$
$h = solveLDE(g,f);$
$print(h);$

Система Mathpar сначала выводит исходную задачу в следующем виде:

$$SPACE = R64[t];$$

$$g = \begin{cases} x''_t + y'_t = sh(t) - sin(t) - t, \\ y''_t + x'_t = ch(t) - cos(t). \end{cases}$$

$$f = \begin{cases} x_{t=0} = 0, \\ x'_{t=0} = 2, \\ y_{t=0} = 1, \\ y'_{t=0} = 0. \end{cases}$$

$$h = solveLDE(g,f);$$

$$print(h);$$

И после этого выводит решение:

$$\begin{cases} x_t = t + 0.5e^t - 0.5e^{-t}, \\ y_t = (-t^2/2) + 0.5e^{it} + 0.5e^{-it}. \end{cases}$$

Пример 2. Решить систему уравнений (24) с начальными условиями (25).

Ввод данных для решения задачи в системе Mathpar

$SPACE = R64[t];$
$g = systLDE(d(x,t) - y + z = 0, -x - y + d(y,t) = 0, -x - z + d(z,t) = 0);$
$f = initCond(d(x,t,0,0) = a, d(y,t,0,0) = b, d(z,t,0,0) = c);$
$h = solveLDE(g,f);$
$print(h);$

Система Mathpar сначала выводит исходную задачу в следующем виде:

$$SPACE = R64[t];$$

$$g = \begin{cases} x'_t - y_t + z_t = 0, \\ -x_t - y_t + y'_t = 0, \\ -x_t - z_t + z'_t = 0. \end{cases}$$

$$f = \begin{cases} x_{t=0} = a, \\ y_{t=0} = b, \\ z_{t=0} = c. \end{cases}$$

$$h = solveLDE(g,f);$$

$$print(h);$$

И после этого выводит решение:

$$\begin{cases} x_t = -b + c + a + (b + (-c))e^t, \\ y_t = -c + (-a) + b + (c + a)e^t + (-c + b)e^t t, \\ z_t = b + (-a) + (-c) + (-b + a + 2c)e^t + (b + (-c))e^t t. \end{cases}$$

9 Заключение

В работе описан алгоритм для нахождения общего и частного решений неоднородной системы обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами. Алгоритм основан на методе преобразования Лапласа, который позволяет свести задачу решения системы обыкновенных линейных дифференциальных уравнений к алгебраической системе уравнений. Разработанный алгоритм позволяет получить как частное, так и общее решение системы дифференциальных уравнений. Решение системы находится в аналитическом виде. Встроен механизм, регулирующий точность получаемого решения. Полученный алгоритм эффективен для решения систем дифференциальных уравнений большого размера.

Разработанный алгоритм был программно реализован. Полученный программный пакет доступен для вычислений в веб-системе Mathpar. Система компьютерной алгебры Mathpar позволяет находить общее и частное решения для системы обыкновенных линейных дифференциальных уравнений с постоянными коэффициентами.

ЛИТЕРАТУРА

1. *Анго А.* Математика для электро- и радиоинженеров. С предисловием Луи Де Бройля. М.: Наука, 1964.

2. *Бояpчук А.К., Головач Г.П.* Справочное пособие по высшей математике. Т.5: Дифференциальные уравнения в примерах и задачах. М.: Эдиториал УРСС, 2001.

3. *Карташев А.П., Рождественский Б.Л.* Обыкновенные дифференциальные уравнения и основы вариационного исчисления. М.: Наука. Главная редакция физико-математической литературы, 1979.

4. *Микусинский Я.* Операторное исчисление. М.: ИЛ, 1956.

5. *Диткин В. А., Прудников А. П.* Интегральные преобразования и операционное исчисление. М.: Физматгиз, 1974.

6. *Шостак Р.Я.* Операционное исчисление. Краткий курс. Изд. второе, доп. Учебное пособие для втузов. М.: Высшая школа, 1972.

7. *Старков В.Н.* Операционное исчисление и его применения. Учебное пособие. СПб.: изд. СПбГУ, 2000.

8. *Ван-дер Поль Б., Бремер Х.* Операционное исчисление на основе двустороннего преобразования Лапласа. М.: ИЛ, 1952.

9. *Дёч Г.* Руководство к практическому применению преобразованию Лапласа. М.: Наука, главная редакция физико-математической литературы, 1965.

10. *Malaschonok N.A.* An Algorithm for Symbolic Solving of Differential Equations and Estimation of Accuracy // Computer Algebra in Scientific Computing. LNCS 5743. Berlin: Springer, 2009. P. 213-225.

11. *Малашонок Г.И., Бетин А.А., Рыбаков М.А., Смирнов Р.А.* Параллельная компьютерная алгебра. Часть 3. Учебное пособие. Тамбов: Издательский дом ТГУ им. Г.Р. Державина, 2012.

12. *Malaschonok G.I.* Project of Parallel Computer Algebra // Tambov University Reports. Series: Natural and Technical Sciences. V. 15. Issue 6. 2010. P. 1724-1729.

13. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 1. 2010. С. 322-327.

14. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 4. 2009. С. 744-748.

15. *Малашонок Г.И.* О вычислении ядра оператора, действующего в модуле // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 13. Вып. 1. 2008. С. 129-131.

16. *Рыбаков М.А.* Решение систем линейных дифференциальных уравнений с кусочно-непрерывными правыми частями с помощью преобразования Лапласа // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 15. Вып. 4. 2010. С. 339-341.

17. *Рыбаков М.А.* Решение систем линейных дифференциальных уравнений с постоянными коэффициентами с помощью преобразования Лапласа // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 4. 2009. С. 791-792.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

COMPUTATION GENERAL AND PARTICULAR SOLUTIONS OF THE INHOMOGENEOUS SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS WITH CONSTANT COEFFICIENTS

© **Mikhail Anatolyevich Rybakov**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Mathematical Analysis Department, e-mail: mixail08101987@mail.ru

Key words: inhomogeneous system of linear ordinary differential equations with constant coe fficients, analytical method of solution, Laplace transform, the algorithm Laplace, Mathpar system.

We discuss the algorithm for obtain the symbolic analytical solution of inhomogeneous system of ordinary linear differential equations with constant coefficients. The developed algorithm may be used for obtaining partial and general solutions of differential equations in an analytical form. It may be used for obtaining the solution with the required accuracy. This algorithm is efficient for the solution of large systems of differential equations. This algorithm is a part of the Mathpar computer algebra system. We demonstrate several examples in the Mathpar system.

УДК 519.688

ПРЕДЕЛ СУММЫ ПОЛОЖИТЕЛЬНЫХ РАЦИОНАЛЬНЫХ СТЕПЕНЕЙ ПОЛИНОМОВ

© **Ю. Ю. Матвеева**

Ключевые слова: полином в рациональной степени, предел суммы полиномов в рациональной степени.

Рассматривается задача вычисления предела суммы положительных рациональных степеней полиномов. Результат формулируется в виде теоремы. Применение теоремы представляет интерес для систем символьных вычислений, в тех случаях, когда выражения содержат сумму полиномов в рациональных степенях.

Определение 1. *Пусть даны полиномы в* $\mathbb{R}[x]$ *:* $P_i(x) = \alpha_{i,n_i}x^{n_i} + \alpha_{i,n_i-1}x^{n_i-1} + \ldots + \alpha_{i,1}x + \alpha_{i,0}$, $\alpha_{i,n_i} \neq 0$, $i = 1, \ldots, k$. *Пусть* λ *– рациональное число,* $d_i = \lambda/n_i$, $i = 1, \ldots, k$, $c_i \in \{1, -1\}$. *Сумму полиномов в рациональных степенях*

$$f = \sum_{i=1}^{k} c_i P_i^{d_i}(x) \tag{1}$$

будем называть однородной суммой степени λ.

Теорема 1. *Пусть* f *есть однородная сумма степени* λ, *заданная формулой* (1). *Введем обозначения для сумм*

$$\phi = \sum_{i=1}^{k} c_i \alpha_{i,n_i}^{d_i}, \quad \phi(s) = \sum_{i=1}^{k} c_i d_i \alpha_{i,n_i}^{d_i-1} \alpha_{i,n_i-s}, \quad s \in \mathbb{N}. \tag{2}$$

Здесь $\alpha_{i,t} = 0$, *если* $t<0$ *или моном со степенью* t *отсутствует в полиноме* $P_i(x)$.

1. Если $\phi \neq 0$, *то* $\lim\limits_{x\to\infty} f = +\infty \cdot \mathbf{sign}(\phi)$.

2. Если $\phi = 0$ *и* $\lambda = 1$, *то* $\lim\limits_{x\to\infty} f = \phi(1)$.

3. Если $\phi = 0$ *и* $\lambda < 1$, *то* $\lim\limits_{x\to\infty} f = 0$.

4. Если $\phi = 0$ *и* $\lambda > 1$, *то обозначим через* m_i *степень следующего после* n_i *ненулевого монома в полиноме* $P_i(x)$, $i = 1, \ldots, k$, $\mu = \min\limits_{i=1,\ldots,k}\{n_i - m_i\}$. *Тогда*

если $\mu > \lambda$, *то* $\lim\limits_{x\to\infty} f = 0$;

если $\mu = \lambda$, *то* $\lim\limits_{x\to\infty} f = \phi(\mu)$;
если $\mu < \lambda$ *и* $\phi(\mu) \neq 0$, *то* $\lim\limits_{x\to\infty} f = +\infty \cdot \mathbf{sign}(\phi(\mu))$, *но если* $\phi(\mu) = 0$, *то требуется дальнейшее исследование.*

Д о к а з а т е л ь с т в о.

Рассмотрим предел для однородной суммы f степени λ.

$$\lim_{x\to\infty} f = \lim_{x\to\infty} \sum_{i=1}^{k} c_i (P_i(x))^{d_i} = \lim_{x\to\infty} \sum_{i=1}^{k} c_i \left(\alpha_{i,n_i} x^{n_i} + \alpha_{i,n_i-1} x^{n_i-1} + \ldots + \alpha_{i,0}\right)^{d_i}.$$

Сделаем замену $x = \frac{1}{y}$. Получим

$$\bar{f}(y) = f\left(\frac{1}{y}\right) = \sum_{i=1}^{k} c_i \frac{(\alpha_{i,n_i} + \alpha_{i,n_i-1} y + \ldots + \alpha_{i,0} y^{n_i})^{d_i}}{y^{n_i d_i}}.$$

Так как $x \to \infty$, то $y \to 0$. Тогда $\lim\limits_{x\to\infty} f(x) = \lim\limits_{y\to 0} \bar{f}(y)$.

Знаменатель каждого слагаемого в $\bar{f}(y)$ равен $y^{n_i d_i} = y^{\lambda}$. Тогда

$$\lim_{y\to 0} \bar{f} = \lim_{y\to 0} \frac{\sum\limits_{i=1}^{k} c_i \left(\alpha_{i,n_i} + \alpha_{i,n_i-1} y + \ldots + \alpha_{i,0} y^{n_i}\right)^{d_i}}{y^{\lambda}}. \tag{3}$$

При $y \to 0$, числитель (3) стремится к сумме коэффициентов старших мономов, т.е. к $\phi = \sum\limits_{i=1}^{k} c_i \alpha_{i,n_i}^{d_i}$, а знаменатель — к 0.

1. **Если** $\phi > 0$, то $\lim\limits_{y\to 0} \bar{f} = +\infty$.

2. **Если** $\phi < 0$, то $\lim\limits_{y\to 0} \bar{f} = -\infty$.

Если $\phi = 0$, то в пределе (3) получим неопределенность $\frac{0}{0}$. В этом случае применим правило Лопиталя.

$$\lim_{y\to 0} \bar{f} =$$

$$\lim_{y\to 0} \frac{\sum_{i=1}^{k} c_i d_i \left(\alpha_{i,n_i} + \alpha_{i,n_i-1} y + \ldots + \alpha_{i,0} y^{n_i}\right)^{d_i-1} \left(\alpha_{i,n_i-1} + 2\alpha_{i,n_i-2} y + \ldots + n_i \alpha_{i,0} y^{n_i-1}\right)}{\lambda y^{\lambda-1}}. \tag{4}$$

3. **Если** $\lambda = 1$, то

$$\lim_{y\to 0} \bar{f} = \sum_{i=1}^{k} c_i d_i \alpha_{i,n_i}^{d_i-1} \alpha_{i,n_i-1}.$$

4. **Если** $0 < \lambda < 1$, то $-\lambda + 1 > 0$. Тогда получим для $\lim_{y\to 0} \bar{f}$ выражение:

$$\lim_{y\to 0} \frac{y^{-\lambda+1}}{\lambda} \sum_{i=1}^{k} c_i d_i (\alpha_{i,n_i} + \alpha_{i,n_i-1} y + \ldots + \alpha_{i,0} y^{n_i})^{d_i-1} (\alpha_{i,n_i-1} + 2\alpha_{i,n_i-2} y + \ldots + n_i \alpha_{i,0} y^{n_i-1}) = 0.$$

5. **Пусть** $\lambda > 1$**.** При $y \to 0$ знаменатель (4) стремится к 0, числитель стремится к числу

$$\beta = \sum_{i=1}^{k} c_i d_i \alpha_{i,n_i}^{d_i-1} \alpha_{i,n_i-1}. \tag{5}$$

Если $\beta > 0$, то $\lim\limits_{y\to 0} \bar{f} = +\infty$.

Если $\beta < 0$, то $\lim\limits_{y\to 0} \bar{f} = -\infty$.

Пусть $\beta = 0$. Обозначим через $s_i = \min\limits_{j=2,\ldots,n_i} \{j : \alpha_{i,n_i-j} \neq 0\}$, $i = 1,\ldots,k$.

Тогда

$$\lim_{y\to 0} \bar{f}(y) = \lim_{y\to 0} \frac{\sum_{i=1}^{k} c_i d_i (\alpha_{i,n_i} + \ldots + \alpha_{i,0} y^{n_i})^{d_i-1} (\alpha_{i,n_i-s_i-1}(s_i+1)y^{s_i} + \ldots + \alpha_{i,0} n_i y^{n_i-1})}{\lambda y^{\lambda-1}}.$$

Пусть $s = \min\limits_{i=1,\ldots,k} \{s_i\}$. Вынесем в числителе функции $\bar{f}(y)$ общий множитель y^s и сократим дробь. Получим

$$\lim_{y\to 0} \bar{f} = \lim_{y\to 0} \frac{y^{s-\lambda+1}}{\lambda} \sum_{i=1}^{k} c_i d_i (\alpha_{i,n_i} + \ldots + \alpha_{i,0} y^{n_i})^{d_i-1} (\alpha_{i,n_i-s_i-1}(s_i+1)y^{s_i-s} + \ldots + \alpha_{i,0} n_i y^{n_i-1-s}).$$

Если $P_i(x) = \alpha_{i,n_i} x^{n_i} + \alpha_{i,m_i} x^{m_i} + \ldots + \alpha_{i,1} x + \alpha_{i,0}$, где $\alpha_{i,n_i} \neq 0$ и $\alpha_{i,m_i} \neq 0$, то

$$P_i(1/y) = \frac{q_i(y)}{y^{n_i}} \quad \text{и} \quad q_i(y) = \alpha_{i,n_i} + \alpha_{i,m_i} y^{n_i-m_i} + \ldots + \alpha_{i,1} y^{n_i-1} + \alpha_{i,0} y^{n_i}.$$

Тогда $q_i'(y) = \alpha_{i,m_i}(n_i - m_i) y^{n_i-m_i-1} + \ldots + \alpha_{i,1}(n_i-1) y^{n_i-2} + \alpha_{i,0} n_i y^{n_i-1}$ и $s_i = n_i - m_i - 1$. Следовательно, $\mu = \min\limits_{i=1,\ldots,k} s_i + 1 = s + 1$ и $s - \lambda + 1 = \mu - \lambda$.

Тогда

$$\lim_{y\to 0} \bar{f} = \lim_{y\to 0} \frac{y^{\mu-\lambda}}{\lambda} \sum_{i=1}^{k} c_i d_i (\alpha_{i,n_i} + \ldots + \alpha_{i,0} y^{n_i})^{d_i-1} (\alpha_{i,n_i-s_i-1}(s_i+1)y^{s_i-s} + \ldots + \alpha_{i,0} n_i y^{n_i-1-s}). \quad (6)$$

Если $\mu > \lambda$, то $\lim\limits_{y\to 0} \bar{f} = 0$.

Если $\mu = \lambda$, то $y^{\mu-\lambda} = 1$. Тогда $\lim\limits_{y\to 0} \bar{f} = \sum\limits_{i=1}^{k} c_i d_i {\alpha_{i,n_i}}^{d_i-1} \alpha_{i,n_i-s-1}$.

Если $\mu < \lambda$, то $y^{\mu-\lambda}$ будет в знаменателе. Тогда обозначим через

$$\gamma = \sum_{i=1}^{k} c_i d_i {\alpha_{i,n_i}}^{d_i-1} \alpha_{i,n_i-s-1}(s+1). \quad (7)$$

При $\gamma > 0$ предел равен $\lim\limits_{y\to 0} \bar{f} = +\infty$, при $\gamma < 0$ предел равен $\lim\limits_{y\to 0} \bar{f} = -\infty$. В случае, если $\gamma = 0$, необходимо дальнейшее исследование, так как снова получаем неопределенность вида $\frac{0}{0}$. Поэтому нужно применить правило Лопиталя к сумме (6).

Т е о р е м а 2. *Пусть* $f_j(x) = \sum\limits_{i=1}^{k_j} c_{ji} P_{ji}^{d_{ji}}(x)$ *есть однородная сумма степени* λ_j, *при этом* $\lambda_j > \lambda_{j+1}$ *для всех* $j = 1,\ldots,m$ *и* $S(x) = \sum\limits_{j=1}^{m} f_j(x)$. *Тогда, если каждый из пределов* $p_j = \lim\limits_{x\to\infty} f_j(x)$ *конечный, то* $\lim\limits_{x\to\infty} S = \sum\limits_{j=1}^{m} p_j$. *Если среди пределов* p_j *есть бесконечные, и* p_t *— это бесконечный предел с наименьшим номером, то* $\lim\limits_{x\to\infty} S = p_t$.

Доказательство.

Первое утверждение теоремы очевидно. Докажем второе.

Не уменьшая общности, будем считать, что все однородные суммы бесконечные, и докажем, что в этом случае предел равен p_1.

Рассмотрим предел суммы $S(x)$

$$\lim_{x\to\infty} S(x) = \lim_{x\to\infty} \sum_{j=1}^{m} f_j(x) = \lim_{x\to\infty} \sum_{j=1}^{m} \sum_{i=1}^{k_j} c_{ji} P_{ji}^{d_{ji}}(x).$$

Сделаем замену переменных, обозначим $x = 1/y$. Получим

$$\lim_{y\to 0} \bar{S}(y) = \lim_{y\to 0} \sum_{j=1}^{m} \sum_{i=1}^{k_j} c_{ji}(\alpha_{j_{i,n_i}} + \alpha_{j_{i,n_i-1}} y + \ldots + \alpha_{j_{i,0}} y^{n_{ji}})^{d_{ji}} / y^{n_{ji} d_{ji}}.$$

Пусть

$$I = \{i : i \in \{1, \ldots, k_1\}; n_{1i} d_{1i} = \lambda_1\}.$$

Приведем дроби к общему знаменателю, получим

$$\lim_{y\to 0} \bar{S}(y) = \lim_{y\to 0} (S_1 + S_2)/y^{\lambda_1},$$

где

$$S_1 = \sum_{i\in I} c_{1i}(\alpha_{1_{i,n_i}} + \alpha_{1_{i,n_i-1}} y + \ldots + \alpha_{1_{i,0}} y^{n_{1i}})^{d_{1i}},$$

$$S_2 = \sum_{j=2}^{m} \sum_{i\notin I} c_{ji}(\alpha_{j_{i,n_i}} + \alpha_{j_{i,n_i-1}} y + \ldots + \alpha_{j_{i,0}} y^{n_{ji}})^{d_{ji}} y^{\lambda_1 - \lambda_j}.$$

При $y \to 0$ предел суммы S_2 равен 0. Следовательно, предел суммы S равен пределу S_1, где S_1 есть сумма $f_1(x)$ после замены $x = \frac{1}{y}$. Теорема доказана.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

LIMIT OF A SUM OF POSITIVE RATIONAL DEGREES OF POLYNOMIALS

© **Julia Yurievna Matveeva**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Student, e-mail: yulya_matveeva91@mail.ru

Key words:rational degrees of polynomials, limit of the sum of rational degrees of polynomials.

An algorithm for finding the limit of a sum of positive rational degrees of polynomials are disccused. Application of the theorem is of interest for symbolic computation systems, in cases where the expression contains the sum of polynomials in the rational powers.

УДК 004.421

ОБ ОДНОМ ПОДХОДЕ К РЕШЕНИЮ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ В ЧАСТНЫХ ПРОИЗВОДНЫХ В ОБЩЕМ ВИДЕ

© **Р. А. Смирнов**

Ключевые слова: дифференциальное уравнение в частных производных, преобразование Лапласа-Карсона, начальные условия, система Mathpar.

В работе рассматривается алгоритм решения дифференциальных уравнений в частных производных в общем виде с применением преобразования Лапласа-Карсона. Приводятся примеры решения таких уравнений в системе Mathpar.

1 Введение

Одной из актуальных задач компьютерной алгебры является задача решения дифференциальных уравнений в частных производных. Данная задача связана с проблемами, возникающими во многих областях естествознания, например, при описании физических процессов таких как нагревание, колебание, диффузия, движение жидкости и многих других.

Известно, что в общее решение обыкновенного дифференциального уравнения входят свободные параметры. Различным наборам значений этих параметров соответствуют различные наборы чисел, определяющих начальные условия. В случае дифференциальных уравнений в частных производных общее решение тоже зависит от свободных параметров. Но теперь эти свободные параметры являются функциями из некоторого класса функций. Особенностью данной работы является то, что предлагается подход, который позволяет в некоторых случаях получить общее решение системы дифференциальных уравнений в частных производных и установить связь свободных функций с начальными условиями, которые задают искомую функцию на границах области.

Работа посвящена решению систем дифференциальных уравнений в частных производных с постоянными коэффициентами, которые допускают преобразование Лапласа-Карсона для функций, стоящих в правой части. Это функции многих переменных $f(x)$, $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, ограниченные, имеющие конечное число точек разрыва I рода и не более чем экспоненциальную скорость роста по каждой переменной на $\mathbb{R}^n_+$ и обращающиеся в 0 в остальных точках $\mathbb{R}^n$. Класс таких функций обозначим через $\mathbf{S_n}$.

Решение основано на преобразовании Лапласа-Карсона и состоит из двух этапов.

Сначала определяются требования к начальным условиям, которые позволяют применить преобразование Лапласа-Карсона к дифференциальным уравнениям и найти изображающие уравнения. Этому посвящен параграф 2. Затем происходит вычисление начальных условий, подстановка изображений начальных условий в изображающее уравнение и нахождение оригинальной искомой функции в результате обратного преобразования Лапласа-Карсона. Этому посвящен параграф 3. В четвертом параграфе приводятся три примера, в которых подробно рассматриваются все этапы полученного алгоритма.

2 Определение требований к начальным условиям

Рассмотрим дифференциальное уравнение в частных производных в $\mathbb{R}^2_+$:

$$\sum_{n=0}^{m} a_n \frac{\partial^m}{\partial x^n \partial y^{m-n}} f(x,y) = h(x,y), \text{ где } a_n \in \mathbb{R},\ h(x,y) \in \mathbf{S_n},\ m \in \mathbb{N}. \tag{1}$$

Прямое преобразование Лапласа-Карсона определяется формулой:

$LC : f(x,y) \mapsto u(p,q) = pq \int\limits_0^\infty \int\limits_0^\infty e^{-px} e^{-qy} f(x,y) dx dy,$

где $p = \sigma + i\mu, q = \tau + i\upsilon$ — комплексные параметры. Известно из работы [1], что:

$$LC : \frac{\partial^n}{\partial x^n} f(x,y) \mapsto p^n u(p,q) - \sum_{k=0}^{n-1} p^{n-k} u_{2,x^k}(0,q),$$

$$LC : \frac{\partial^n}{\partial y^n} f(x,y) \mapsto q^n u(p,q) - \sum_{k=0}^{n-1} q^{n-k} u_{1,y^k}(p,0),$$

$$LC : \frac{\partial^{m+n}}{\partial x^m \partial y^n} f(x,y) \mapsto p^m q^n u(p,q) - p^m \sum_{l=0}^{n-1} q^{n-l} u_{1,y^l}(p,0) - q^n \sum_{k=0}^{m-1} p^{m-k} u_{2,x^k}(0,q) +$$

$$+ \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} p^{m-k} q^{n-l} f^{(k+l)}_{x^k y^l}(0,0),$$

где $m, n \geqslant 1; u_{2,x^k}(0,q) = q \int\limits_0^\infty e^{-q\eta} \left.\frac{\partial^k f(x,\eta)}{\partial x^k}\right|_{x=0} d\eta;\ u_{1,y^k}(p,0) = p \int\limits_0^\infty e^{-q\xi} \left.\frac{\partial^k f(\xi,y)}{\partial y^k}\right|_{y=0} d\xi.$

Таким образом, после преобразования Лапласа-Карсона левой части уравнения (1) нам необходимо знать значения следующих функций, которые называются начальными условиями и их можно задать в виде:

$$\left.\frac{\partial^k f(x,y)}{\partial x^k}\right|_{x=0} = a_k(y),\ \left.\frac{\partial^h f(x,y)}{\partial y^h}\right|_{y=0} = b_h(x), \tag{2}$$

где $k = 0, \ldots, n-1$; n — порядок производной функции f по переменной x в (1); $h = 0, \ldots, m-1$; m — порядок производной функции f по переменной y в (1).

3 Определение семейства начальных условий

Пусть искомая функция $f = f(x,y)$ и в уравнении (1) не содержится смешанных производных функции $f(x,y)$, k — старшая производная функции f по переменной x, m — старшая производная функции f по переменной y.

Пусть заданы начальные условия

$$IC = \{a_0, \ldots, a_{k-1}, b_0, \ldots, b_{m-1}\},$$

где $a_i = \left.\dfrac{\partial^i f(x,y)}{\partial x^i}\right|_{x=0}$, $b_j = \left.\dfrac{\partial^j f(x,y)}{\partial y^j}\right|_{y=0}$; $i = 0, \ldots, k-1$; $j = 0, \ldots, m-1, a_i, b_j \in \mathbf{S_2}$.

В результате преобразования Лапласа-Карсона начальных условий получим

$$LC : f(x,y) \mapsto u(p,q);\ a_i(y) \mapsto \alpha_i(q);\ b_j(x) \mapsto \beta_j(p).$$

После преобразования Лапласа-Карсона и решения полученного алгебраического уравнения получим изображение u искомой функции f в виде дробно-рациональной функции:

$$u = \frac{\Omega(p,q)}{Q(p,q)}, \tag{3}$$

где $\Omega(p,q) = \sum\limits_{i=0}^{k-1} P_i(p,q)\alpha_i + \sum\limits_{j=0}^{m-1} P_j(p,q)\beta_j + \Theta(p,q)$.

Пусть знаменатель $Q(p,q)$ можно представить виде $Q(p,q) = \prod\limits_{r=1}^{l}(p - \psi_r(q))$, где $l \in \mathbb{N}$, причём, $\psi_i(q) \neq \psi_j(q)$ при $i \neq j$, $i, j = 1, \ldots, l$.

Обратное преобразование Лапласа-Карсона для двух переменных задается формулой:

$$LC^{-1} : u(p,q) \mapsto f(x,y) = \int\limits_{\sigma-i\infty}^{\sigma+i\infty}\int\limits_{\tau-i\infty}^{\tau+i\infty} \frac{u(p,q)}{pq} e^{px+qy} dpdq,$$

где $p = \sigma + i\mu$, $q = \tau + i\upsilon$ — комплексные параметры.

Для вычисления оригинала f с помощью применения обратного преобразования Лапласа-Карсона к изображению u, необходимо, чтобы существовали m, $n \in \mathbb{R}$ такие, что выполняется неравенство $|u| < \infty$ при любых $Re(p) > n > 0$, $Re(q) > m > 0$. Выражение $p - \psi_r(q)$ обращается в ноль в области $Re(p) > n > 0$, $Re(q) > m > 0$, m, $n \in \mathbb{R}$, тогда и только тогда, когда функция $\psi_r(q)$ удовлетворяет следующим условиям

$$\begin{cases} \lim\limits_{\rho\to+\infty} |\psi_r(q)| = +\infty, \\ \pi/2 \leqslant Arg(\lim\limits_{\rho\to+\infty}(\psi_r(q), q)) \leqslant 2\pi, \end{cases} \tag{4}$$

где $q = \rho e^{i\varphi}$, $0 < \varphi < \pi/2$, $r = 0, \ldots, l$.

Следовательно, такие функции $\psi_r(q)$, которые удовлетворяют системе (4), должны входить как сомножители в разложение числителя (3).

Запишем $Q(p,q)$ в виде $Q(p,q) = Q_1(p,q) * Q_2(p,q)$, где $Q_1(p,q) = \prod\limits_{c=1}^{d}(p - \psi_c(q))$, $Q_2(p,q) = \prod\limits_{e=d+1}^{l}(p - \psi_e(q))$, $\psi_c(q)$ удовлетворяют (4), а $\psi_e(q)$ не удовлетворяют (4). Подставляя функции $\psi_c(q)$ в $\Omega(p,q)$, получим d уравнений:

$$\begin{cases} \Omega(\psi_1(q), q) = 0, \\ \ldots \\ \Omega(\psi_d(q), q) = 0. \end{cases} \tag{5}$$

Заметим, что α_i и β_j входят в Ω линейно, поэтому (5) является неоднородной системой линейных уравнений относительно функций α_i и β_j.

Для того чтобы функции a_i, b_j могли выступать как начальные условия решения дифференциального уравнения в частных производных необходимо и достаточно, чтобы они удовлетворяли системе уравнений (5). Покажем достаточность. Функция $u = \dfrac{\Omega(p,q)}{Q(p,q)}$ — дробно-рациональная функция от p и q, для которой существуют $m,\ n \in \mathbb{R}$ такие, что выполняется $Q(p,q) \neq 0$ при любых $Re(p) > n > 0,\ Re(q) > m > 0$.

Пусть T есть основная матрица коэффициентов системы (5) и $rank(T) = t$, причём $t \leqslant d \leqslant k+m-2$. Тогда из системы (5) мы можем выразить t неизвестных как линейные комбинации остальных $k+m-2-t$ свободных неизвестных, при этом коэффициентами будут дробно-рациональные функции.

В качестве свободных переменных могут выступать любые функции из класса $\mathbf{S_n}$. Эти свободные функции перейдут и в искомое решение дифференциального уравнения (1) в общем виде.

4 Примеры

Пример 1. Решим уравнение в частных производных в $\mathbb{R}^2_+$.

$$\frac{\partial f}{\partial x} - \frac{\partial f}{\partial y} = xy, \text{ где} f = f(x,y). \tag{6}$$

Запишем начальные условия в соответствии с формулой (2):

$$f(0,y) = a(y);\ f(x,0) = b(x).$$

После преобразования по Лапласу-Карсону функции и начальных условий получаем:

$$f(x,y) \mapsto u(p,q);\ a(y) \mapsto \alpha(q);\ b(x) \mapsto \beta(p),$$

где $p = \mu + i\kappa,\ q = \eta + i\sigma$.

Изображающее уравнение с подставленными начальными условиями будет иметь вид:

$$pu - qu = \frac{1}{pq} + \alpha(q)p - \beta(p)q.$$

Отсюда выразим функцию u:

$$u = \frac{\alpha(q)p^2q - \beta(p)pq^2 + 1}{pq(p-q)}. \tag{7}$$

Знаменатель обращается в ноль при $p - q = 0$, следовательно,

$$\alpha(q)p^2q - \beta(p)pq^2 + 1 = 0 \text{ или } \alpha(q) - \beta(q) + \frac{1}{q^3} = 0. \tag{8}$$

После обратного преобразования Лапласа-Карсона уравнения (8) имеем следующую связь между начальными условиями:

$$LC^{-1}(\alpha(q)) - LC^{-1}(\beta(q)) + LC^{-1}(\frac{1}{q^3}) = 0.$$

Пусть $LC^{-1}(\alpha(q)) = W(y), LC^{-1}(\beta(p)) = U(x)$. Тогда $U(x) = W(x) + \frac{x^3}{6}$, где $W(x)$ — произвольная функция из $\mathbf{S_1}$.

Следовательно, искомые семейства начальных условий можно записать в виде:

$$a(y) = W(y), b(x) = W(x) + x^3/6.$$

Решим уравнение (6) в общем виде. Выразим $\beta(p)$ из уравнения (8) и подставим полученное выражение в (7). Получим:

$$u = \frac{\alpha(p)p - \alpha(q)q}{p - q} + \frac{p + q}{p^3 q}.$$

После обратного преобразования Лапласа-Карсона получаем решение уравнения (6) в общем виде:

$$u = W(x + y) + \frac{x^2 y}{2} + \frac{x^3}{6}.$$

Пример 2. Решим уравнение параболического вида, приведенное в работе [2].

$$\frac{\partial^2 f}{\partial x^2} - \frac{\partial f}{\partial y} = xy, \text{ где } f = f(x, y). \tag{9}$$

Запишем начальные условия в соответствии с формулой (2):

$$f(0, y) = a(y); \ \left.\frac{\partial f(x, y)}{\partial x}\right|_{x=0} = b(y); \ f(x, 0) = c(x).$$

После преобразования по Лапласу-Карсону функции и начальных условий получаем:

$$f(x, y) \mapsto u(p, q); \ a(y) \mapsto \alpha(q); \ b(y) \mapsto \beta(q); \ c(x) \mapsto \gamma(p),$$

где $p = \mu + i\kappa, \ q = \eta + i\sigma$.

Изображающее уравнение с подставленными начальными условиями будет иметь вид

$$p^2 u - qu = \frac{1}{pq} + \alpha(q)p^2 + \beta(q)p - \gamma(p)q.$$

Выразим функцию u

$$u = \frac{\alpha(q)p^3 q + \beta(q)p^2 q - \gamma(p)pq^2 + 1}{pq(p^2 - q)}. \tag{10}$$

Знаменатель имеет ноль $q = p^2$. Сделаем подстановку $q = p^2$. Получим следующую зависимость между начальными условиями:

$$1 + p^5\alpha(p^2) + p^4\beta(p^2) - p^5\gamma(p) = 0. \tag{11}$$

Пусть $LC^{-1}(\alpha(q)) = W(y), \ LC^{-1}(\beta(q)) = U(y)$, где $W(t), \ U(t)$ — произвольные функции из $\mathbf{S_1}$.

Выразим функцию $\gamma(p)$ из уравнения (11) и сделаем подстановку в (10):

$$u = \frac{p^2\alpha(q) - q\alpha(p^2)}{p^2 - q} + \frac{p^2\beta(q) - q\beta(p^2)}{p(p^2 - q)} + \frac{p^2 + q}{p^5 q}.$$

После обратного преобразования получаем символьное решение уравнения (9) в общем виде:

$$\begin{cases} W(y) = LC^{-1}(\alpha(q)), \quad U(y) = LC^{-1}(\beta(q)), \\ f = LC^{-1}(\frac{p^2\alpha(q) - q\alpha(p^2)}{p^2 - q} + \frac{p^2\beta(q) - q\beta(p^2)}{p(p^2 - q)}) + \frac{x^3 y}{6} + \frac{x^5}{120}. \end{cases}$$

Пример 3. Дано уравнение в частных производных в $\mathbb{R}^2_+$

$$\frac{\partial^2 f}{\partial x^2} - \frac{\partial^2 f}{\partial y^2} = xy, \text{ где } f = f(x, y). \tag{12}$$

Запишем начальные условия в соответствии с формулой (2):

$$f(0, y) = a(y); \; f(x, 0) = b(x); \; \left.\frac{\partial f(x, y)}{\partial x}\right|_{x=0} = c(y); \; \left.\frac{\partial f(x, y)}{\partial y}\right|_{y=0} = d(x).$$

После преобразования по Лапласу-Карсону функции и начальных условий получаем

$$f(x, y) \mapsto u(p, q); \; a(y) \mapsto \alpha(q); \; b(x) \mapsto \beta(p); \; c(y) \mapsto \gamma(q); \; d(x) \mapsto \delta(p),$$

где $p = \mu + i\kappa, \; q = \eta + i\sigma$.

Изображающее уравнение с подставленными начальными условиями будет иметь вид:

$$p^2 u - q^2 u = \frac{1}{pq} + p^2\alpha(q) + p\gamma(q) - q^2\beta(p) - q\delta(p).$$

Выразим функцию u

$$u = \frac{p^3 q\alpha(q) + p^2 q\gamma(q) - pq^3\beta(p) - pq^2\delta(p) + 1}{pq(p^2 - q^2)}. \tag{13}$$

Знаменатель обращается в ноль при $p = q$, следовательно,

$$q^4\alpha(q) + q^3\gamma(q) - q^4\beta(p) - q^3\delta(p) + 1 = 0. \tag{14}$$

После обратного преобразования Лапласа-Карсона уравнения (14) имеем следующую связь между начальными условиями:

$$LC^{-1}(q\alpha(q)) - LC^{-1}(q\beta(p)) + LC^{-1}(\gamma(q)) - LC^{-1}(\delta(p)) + LC^{-1}\left(1/q^3\right) = 0.$$

Пусть $LC^{-1}(\alpha(q)) = W(y), LC^{-1}(\beta(p)) = U(x), \; LC^{-1}(\gamma(q)) = V(y)$, где $W(t), U(t), V(t)$ — произвольные функции из $\mathbf{S_1}$.

Выразим функцию $\delta(p)$ и подставим в уравнение (13):

$$u = \frac{1}{p^3 q} + \frac{p\gamma(q) - q\gamma(p)}{p^2 - q^2} + \frac{p^2\alpha(q) - pq\alpha(p)}{p^2 - q^2} + \frac{p\beta(p)}{p + q}.$$

После обратного преобразования получаем символьное решение уравнения (12) в общем виде:

$$\begin{cases} W(y) = LC^{-1}(\alpha(q)), \; U(x) = LC^{-1}(\beta(p)), \; V(y) = LC^{-1}(\gamma(q)), \\ f = LC^{-1}(\frac{p\gamma(q) - q\gamma(p)}{p^2 - q^2} + \frac{p^2\alpha(q) - pq\alpha(p)}{p^2 - q^2} + \frac{p\beta(p)}{p + q}) + \frac{x^3 y}{6}. \end{cases}$$

5 Заключение

В работе показан один подход к решению дифференциальных уравнений в частных производных в общем виде. В настоящее время на его основе разрабатывается алгоритм в системе компьютерной алгебры Mathpar. Данный метод позволяет определить требования к начальным условиям, при которых возможно найти решение при помощи преобразования Лапласа-Карсона.

ЛИТЕРАТУРА

1. *Диткин В.А., Прудников А.П.* Операционное исчисление по двум переменным и его приложения. М.: Физматгиз, 1958.

2. *Малашонок Н.А.* Один пример символьного решения системы дифференциальных уравнений в частных производных // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 15. Вып. 6. 2010. С. 1761-1766.

3. *Malaschonok N.A.* An Algorithm for Symbolic Solving of Differential Equations and Estimation of Accuracy // Computer Algebra in Scientific Computing. LNCS 5743. Springer, Berlin. 2009. P. 213-225.

4. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 4. 2009. С. 744-748.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

SYMBOLIC SOLUTION OF PARTIAL DIFFERENTIAL EQUATIONS IN GENERAL FORM

© **Roman Antonovich Smirnov**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Mathematical Analysis Department, e-mail: romansmirnovtsu@gmail.com

Key words: partial differential equations; Laplas-Carson transform; initial conditions, Mathpar system.

In this paper the algorithm for symbolic solution of partial differential equations in general form is discribe. Examples of solutions of these equations are demonstrated in the Mathpar system.

УДК 519.688

ОБ ОДНОМ МАТРИЧНОМ ПОДХОДЕ К ПОСТРОЕНИЮ БАЗИСОВ ГРЕБНЕРА

© **И. А. Борисов**

Ключевые слова: базисы Грёбнера, матричная редукция, система компьютерной алгебры Mathpar.

Мы рассматриваем матричный подход к построению базисов Грёбнера полиномиальных идеалов и новые структуры данных для эффективной организации вычислений в системе компьютерной алгебры Mathpar.

1 Введение

Базисы Гребнера полиномиальных идеалов имеют широкое применение. Существуют различные алгоритмы построения базисов Гребнера [1], [2]. Матричные алгоритмы и способы их распареллеливания приводятся в статьях [3], [4], [5].

В данной статье рассматриваются новые структуры данных, которые разработаны специально для матричного подхода к вычислению базисов полиномиальных идеалов.

В параграфе 2 вводятся основные определения. В параграфе 3 описывается матричный алгоритм Фужера `F4`. В параграфе 4 предлагаются структуры данных для хранения полиномов, которые приспособлены для матричных алгоритмов, используемых при вычислении полиномиальных базисов. В параграфе 5 приводится пример с использованием этих структур данных.

2 Базисы Гребнера

Определение 1. *Пусть R — коммутативное кольцо. Подмножество $I \subset R$ называется* идеалом, *если выполнены следующие условия [2]: 1) $0 \in I$; 2) если $a, b \in I$, то $a + b \in I$; 3) если $a \in I, b \in R$, то $a \cdot b \in I$.*

Если идеал I порождён полиномами $g_1, g_2, \ldots, g_n$, то множество $G = \{g_1, g_2, \ldots, g_n\}$ называется *базисом* (системой образующих) идеала I ($I = \langle G \rangle$).

Определение 2. *Идеал $I \subset k[x_1, \ldots, x_n]$ называется* мономиальным *($\langle x^\alpha : \alpha \in A \rangle$), если существует подмножество $A \subset Z^n_{\geqslant 0}$ такое, что I состоит из всех конечных сумм вида $\sum_{\alpha \in A} h_\alpha x^\alpha$, где $h_\alpha \in k[x_1, \ldots, x_n]$ [2].*

Определение 3. Мономиальным упорядочением *на* $k[x_1, \ldots, x_n]$ *называется бинарное отношение* $>$ *на множестве* $Z^n_{\geqslant 0}$, *обладающее следующими свойствами [2]: 1)* $>$ *является линейным упорядочением на* $Z^n_{\geqslant 0}$; *2) если* $\alpha > \beta$ *и* $\gamma \in Z^n_{\geqslant 0}$, *то* $\alpha+\gamma > \beta+\gamma$; *3)* $>$ *вполне упорядочивает* Z^n_0.

Относительно выбранного упорядочения будем пользоваться следующими обозначениями: 1) $HT(f)$ — *старший терм, т.е. старший моном без числового коэффициента*; 2) $HC(f)$ — *старший коэффициент*; 3) $HM(f)$ — *старший моном, т.е.* $HM(f) = HC(f)HT(f)$.

Определение 4. *Полином* f редуцируется *к полиному* h *по модулю* G, *если в* G *существует некоторый полином* g, *такой что* $HM(f) = HM(g) \cdot m$, *где* $m \in R$ *и* $h = f - g \cdot m$. Будем обозначать $f \underset{G}{\to} h$.

Определение 5. *Множество* $G \subset I$ *называется* базисом Гребнера *идеала* I, *если идеал, порождённый множеством старших мономов полиномов* G, *совпадает с идеалом, порожденным старшими мономами* I: $\langle HM(G)\rangle = \langle HM(I)\rangle$.

Для того, чтобы множество G из идеала I являлось базисом Гребнера, необходимо и достаточно, чтобы любой полином из I редуцировался по модулю G [3].

Б. Бухбергер доказал, что можно проверять редукцию не всех $f \in I$, а только его некоторого подмножества и сформулировал алгоритмическое определение базиса Гребнера [4].

Определение 6. *S-полиномом, порождённым* f *и* g, *называется полином вида:* $S(f,g) = \frac{LCM(HT(f),HG(g))}{LM(f)} \cdot f - \frac{LCM(HT(f),HT(g))}{LM(g)} \cdot g$.

Теорема 1 (Бухбергера). *Если для всех пар* $(f,g) \in G$ *их S-полиномы редуцируются к нулю по модулю* G, *то* G *является базисом Гребнера.*

Эта теорема лежит в основе алгоритма Бухбергера.

3 Матричный алгоритм вычисления базисов Гребнера

В 1999 г. Ж. К. Фужер представил эффективный матричный алгоритм построения базисов Гребнера F_4 [5].

Основные отличия от алгоритма Бухбергера: 1) Представление полиномов: вместо множества полиномов используется матрица коэффициентов этих полиномов. 2) Составление S-полиномов: алгоритм Фужера позволяет не вычислять S-полиномы полностью, а добавлять в матрицу их части для дальнейшего составления линейных комбинаций. 3) Редукция S-полиномов: редукция выполняется за один шаг с помощью приведения матрицы коэффициентов полиномов к ступенчатому виду. Чтобы эта операция была эквивалентна редукции, в матрицу добавляются строки, которые соответствуют полиномам базиса с некоторыми множителями.

Для построения матрицы необходимо задать мономиальное упорядочение и поставить в соответствие каждому моному каждого полинома коэффициент матрицы.

Пример соответствия матрицы множеству полиномов приведён на рис. 1.

На рис. 2, 3, 4 приведён алгоритм Фужера.

	z^3	z	y^2	yx^2
p_1	0	2	0	1
p_2	1	0	-3	0
p_3	-10	0	0	0

Рис. 1: Соответствие матрицы множеству полиномов $\{ p_1 = 2z + yx^2, p_2 = z^3 - 3y^2, p_3 = -10z \}$

```
function F4( F )                                  ▷ F — исходный идеал
    G = F
    P = {(f, g) : f, g ∈ G, f ≠ g}                ▷ Получение всех пар из G
    while P ≠ ∅ do
        Sel = Select(P)                           ▷ Выбор и удаление критических пар
        LR = {(left, right) : p, p ∈ Sel}         ▷ Получение левых и правых частей
S-полиномов пар
        reduced = reduction(LR, G)
        for all r ∈ reduced do
            if HT(r) ∉ HT(G) then
                for all g ∈ G do                  ▷ Обновление списка пар
                    P = P ∪ (r, g)
                end for
            end if
        end for
    end while
    return G
end function
```

Рис. 2: Алгоритм F4

4 Структуры данных для матричного алгоритма

Существующая реализация содержит преобразование множества полиномов в матрицу коэффициентов и восстановление множества полиномов из матрицы коэффициентов и набора термов на каждой итерации вычисления базиса Гребнера. Данный подход является неэффективным, так как при каждой операции получения матрицы коэффициентов из множества полиномов выполняется выделение и заполнение двумерных массивов большого размера, и затем при восстановлении полиномов из матрицы производится поиск термов, нужных для получения каждого полинома, из списка всех термов всех полиномов базиса.

Для ускорения матричного алгоритма предлагается специальная структура данных, которая позволяет не совершать упомянутые выше преобразования, а производит все необходимые действия только над матрицей коэффициентов. Она описана в классе `PolynomialList`.

Класс `PolynomialList` содержит методы, требуемые на различных этапах исполнения матричного алгоритма. Основными операциями являются: создание объекта `PolynomialList`

function REDUCTION((LR, G))
  $PP =$ **symPP**(LR, G) ▷ препроцессинг
  $reduced = row\ echelon\ form\ w.r.t.\ PP$ ▷ преобразовываем полиномы в матрицу, приводим ее к ступенчатому виду и восстанавливаем полиномы
  for all $r \in reduced$ **do**
    if $HT(r) \notin PP$ **then** ▷ возвращаем только редуцированные полиномы
      $res = res \cup r$
    end if
  end for
  return res
end function

Рис. 3: Функция редукции

function SYMPP(LR, G)
  $F = \{t * f, (t, f) \in LR\}$
  $Done = HT(F)$
  while $Done \neq T(F)$ **do**
    $Done = Done \cup \{t, t \in T(F)/Done\}$ ▷ выбираем терм t
    if $t\ is\ top\ reducible\ by\ G$ **then** ▷ если t редуцируется каким-либо полиномом из G
      $r = reductor\ of\ t$
      $m = \frac{t}{HT(r)}$
      $res = res \cup \{m \cdot r\}$
    end if
  end while
  return res
end function

Рис. 4: Функция построения матрицы

из массива полиномов, получение и выбор критических пар, умножение терма на полином и добавление полинома в существующий объект. Главные методы класса приведены на рис. 5, 6, 7.

Множество полиномов представляется следующим образом: все термы всех полиномов хранятся в сортированном списке (`AllTerms` типа `TermList`); сами полиномы — в строках двух двумерных массивов, один из которых содержит индексы термов в списке (`ColIndexes` — массив целых чисел), а другой хранит соответствующие коэффициенты (`Rows` — массив типа `Element`).

Термы помещаются в специальный объект класса `TermList`, который обеспечивает постоянное упорядочение термов по убыванию и обновление индексов существующих термов при добавлении новых.

Во время выполнения матричного алгоритма существуют три объекта `PolynomialList`. Первый объект используется для хранения текущего состояния базиса. Второй объект содержит левые и правые части S-полиномов критических пар. Третий объект служит для подготовки к редукции, в него помещаются выбранные на данном шаге критические

function PolynomialList.add(p)
$T = getAllTerms(p)$ ▷ Получаем все термы.
$NewIndexes, UpdatedIndexes = AllTerms.addAll(T)$ ▷ Добавляем термы в список всех термов, получая индексы новых элементов и обновленные индексы.
$Rows.add(getAllCoeffs(p))$ ▷ Добавляем новую строку с коэффициентами полинома.
$ColIndexes.add(NewIndexes)$ ▷ Добавляем новую строку с индексами термов.
$updateIndexes(UpdatedIndexes)$ ▷ Обновляем изменившиеся индексы термов.
end function

Рис. 5: PolynomialList. Добавление полинома

function PolynomialList.multOnTerm(polIndex, t)
for all $currTerm = AllTerms[ColIndexes_i]$ **do** ▷ Умножаем все термы полинома на данный терм.
$NewIndex, UpdatedIndexes = AllTerms.add(currTerm * t)$
$updateIndexes(UpdatedIndexes)$
end for
end function

Рис. 6: PolynomialList. Умножение полинома на терм

function TermList.add(t)
$newIndex = getInsertIndex(t)$ ▷ Получаем индекс для вставки (двоичный поиск).
$insert(t, newIndex)$
$updatedIndexes = updateIndexes(newIndex)$ ▷ Обновляем индексы термов, младших t.
return $(newIndex, updatedIndexes)$
end function

Рис. 7: TermList. Добавление терма

пары и полиномы, полученные в результате выполнения препроцессинга.

5 Примеры

В примерах используется обратный лексикографический порядок $z > y > x$.

На рис. 8 приведена схема представления множества полиномов $\{ p_1 = 2z + yx^2, p_2 = z^3 - 3y^2, p_3 = -10z \}$ в виде объекта `PolynomialList`.

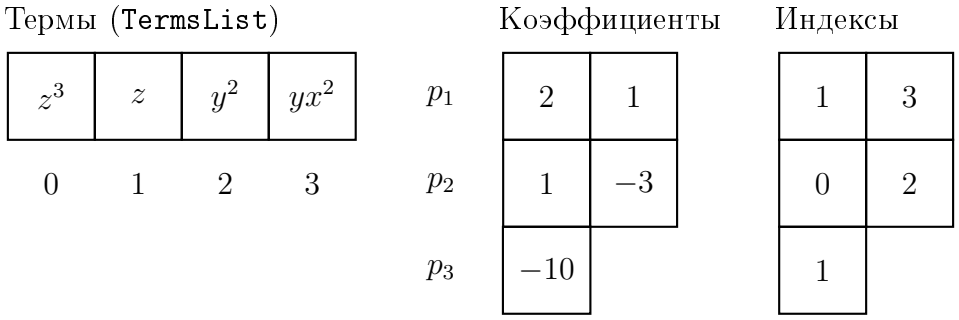

Рис. 8: Представление множества полиномов в виде `PolynomialList`

На рис. 9 показано состояние объекта после добавления нового полинома $p_4 = 5z^2 - 2y^2$, на рис. 10 — после умножения полинома p_3 на терм x^2.

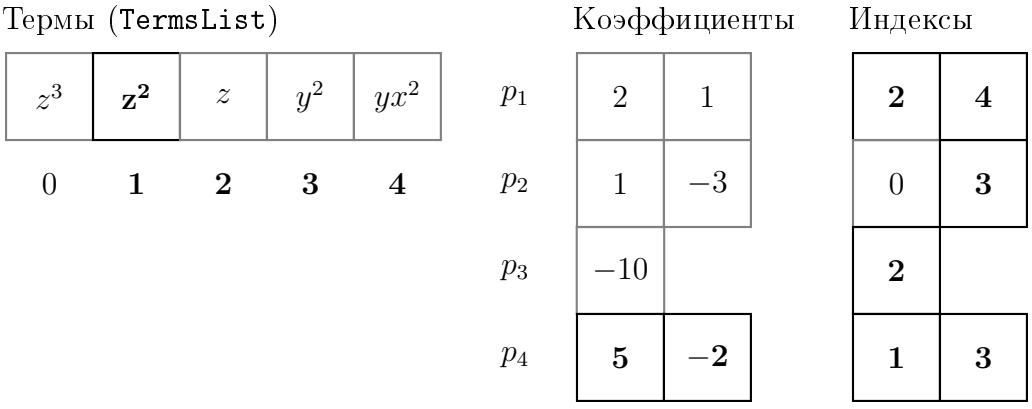

Рис. 9: Добавление полинома в объект `PolynomialList`

6 Заключение

В дальнейшем планируется развитие матричного алгоритма и его распареллеливание за счет матричных операций для исполнения на параллельных вычислительных машинах с распределенной памятью.

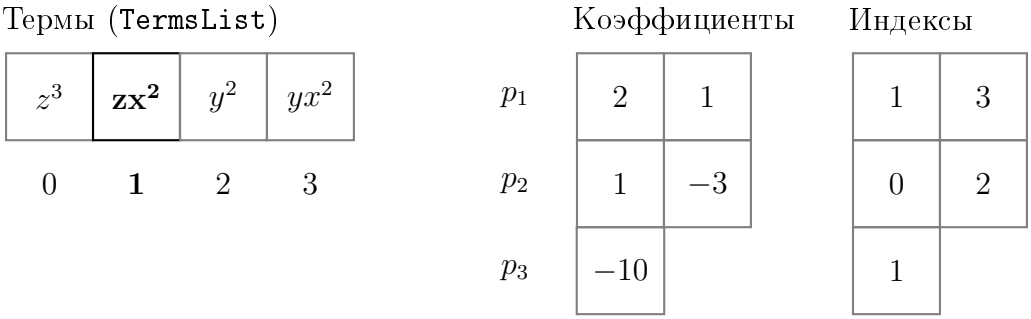

Рис. 10: Умножение полинома на терм

ЛИТЕРАТУРА

1. *Малашонок Г.И., Борисов И.А.* Некоторые подходы к вычислению базисов Гребнера // Вестник Тамбовского Университета. Исследовательские проекты студентов. Приложение к журналу. 2011. С. 191-194.

2. *Кокс Д., Литтл Дж., О'Ши Д.* Идеалы, многообразия и алгоритмы. М.: Мир, 2000.

3. *Becker T., Wespfenning V.* Groebner bases. A computational approach to commutative arlgebra. New York: Springer, 1993. V. 141.

4. Компьютерная алгебра: символьные и алгебраические вычисления / пер. с англ. *под ред. Б. Бухбергера, Дж. Колинза, Р. Лооса.* М.: Мир, 1986.

5. *Faugere J.-C.* A new efficient algorithm for computing Groebner bases (F4) // J. Pure Appl. Algebra. 1999. V. 139. No 1–3. P. 61–88.

6. *Малашонок Г.И., Старов М.В., Борисов И.А.* К параллельному вычислению базисов Гребнера // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов. Т. 15. Вып. 6. 2010. С. 1755-1760.

7. *Старов М.В.* Реализация метода Фужера // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов. Т. 14. Вып. 4. 2009. С. 802-803.

8. *Александров Д.Е., Галкин В.В., Зобнин А.И., Левин М.В.* Распараллеливание матричных алгоритмов вычисления базисов Гребнера // Фундаментальная и прикладная математика. Москва. Т. 14. Вып. 4. 2008. С. 35-64.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

ABOUT ONE MATRIX APPROACH TO CONSTRUCTING GRÖBNER BASES

© **Ivan Andreevich Borisov**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Mathematical Analysis Department, e-mail: ivan@iborisov.ru

Key words: Gröbner bases, matrix reduction, Mathpar computer algebra system.

We describe matrix approach to constructing Gröbner bases of polynomial ideals and data structures needed to efficient execution in computer algebra system Mathpar.

УДК 519.688

ПАРАЛЛЕЛЬНЫЙ МОДУЛЯРНЫЙ АЛГОРИТМ ВЫЧИСЛЕНИЯ ПРИСОЕДИНЁННОЙ МАТРИЦЫ В КОЛЬЦЕ ПОЛИНОМОВ МНОГИХ ПЕРЕМЕННЫХ

© А. А. Бетин

Ключевые слова: вычисление присоединённой матрицы, параллельный алгоритм, система Mathpar.

Приводится параллельный модулярный алгоритм вычисления присоединённой матрицы в кольце полиномов многих переменных, вычисляется верхняя оценка наибольшего по абсолютной величине числового коэффициента элементов присоединённой матрицы. Обсуждается реализация алгоритмов в системе Mathpar.

1 Введение

Вычисление присоединённой матрицы в кольце полиномов – это вычислительно сложная задача, так как с ростом размеров матрицы происходит быстрый рост коэффициентов и степеней полиномов, которые являются элементами присоединенной матрицы. Уменьшить вычислительную сложность позволяет применение китайской теоремы об остатках (КТО). Такой подход позволяет прийти к большому числу независимых задач. Это обеспечивает хороший параллелизм алгоритма. Присоединенная матрица каждый раз вычисляется в некотором конечном поле. Затем искомый результат получается в результате применения интерполяционных формул с использованием схем Ньютона или Лагранжа.

Для использования КТО необходимо выбрать достаточное количество модулей, а для этого требуется найти верхнюю оценку результата. В параграфе 2 вычисляется оценка максимального по абсолютной величине числового коэффициента элемента присоединённой матрицы.

В параграфе 3 даётся параллельный алгоритм вычисления присоединённой матрицы, который использует вычисления в конечных полях.

2 Оценки для элементов присоединённой матрицы

Пусть $f,\ g \in Z[x_0, \ldots, x_l]$, f содержит m ненулевых мономов, g содержит k ненулевых мономов. Обозначим через $maxCoeff(p)$ максимальный по абсолютной величине

числовой коэффициент полинома p. Тогда максимальный по абсолютной величине числовой коэффициент в произведении полиномов f и g удовлетворяет неравенству

$$maxCoeff(fg) \leqslant maxCoeff(f) \cdot maxCoeff(g) \cdot min\{m, k\}. \tag{1}$$

Пусть дана матрица A размера $n \times n$ с элементами из кольца полиномов $a_{ij} \in Z[x_0, x_1, \ldots, x_l]$. Обозначим через $\tilde{A}$ присоединённую матрицу для матрицы A. Тогда $\tilde{A} = (A_{ij})$, где A_{ij} — алгебраическое дополнение элемента a_{ij} матрицы A, т.е. $A_{ij} = (-1)^{i+j} M_{ij}$, где M_{ij} — это минор элемента a_{ij}, полученный из определителя матрицы A вычеркиванием i-ой строки и j-го столбца, $i, j = 1, \ldots, n$. Обозначим через $\tilde{a_{ij}}$ элемент присоединенной матрицы, $i, j = 1, \ldots, n$ и найдем оценку для максимального по абсолютной величине числового коэффициента у полиномов, которые являются элементами присоединенной матрицы.

Определитель матрицы A можно вычислить по формуле

$$\det A = \sum_{(j_1, \ldots, j_n)} (-1)^t a_{1j_1} a_{2j_2} \cdots a_{nj_n}, \tag{2}$$

где сумма берется по всем перестановкам чисел от 1 до n: $(j_1, \ldots, j_n)$, $t = t(j_1, \ldots, j_n)$ — четность этой перестановки.

Если A — полиномиальная матрица, то определитель $detA$, является суммой произведений полиномов. В соответствии с (1), получаем

$$maxCoeff(det(A)) \leqslant \sum_{(j_1, \ldots, j_n)} C_{1j_1} \cdot C_{2j_2} \cdots C_{nj_n} \cdot p_{2j_2} \cdot p_{3j_3} \cdots p_{nj_n},$$

где C_{nj_n} — максимальный по абсолютной величине числовой коэффициент полинома a_{nj_n}, p_{nj_n} — число мономов в полиноме a_{nj_n}.

Выбрав из каждой строки матрицы максимальный по абсолютному значению числовой коэффициент C_{max_i}, $i = 1, \ldots, n$, и максимальное количество мономов p_{max_i}, $i = 2, \ldots, n$, получим:

$$\sum_{(j_1, \ldots, j_n)} C_{1j_1} \cdot C_{2j_2} \cdots C_{nj_n} \cdot p_{2j_2} \cdot p_{3j_3} \cdots p_{nj_n} \leqslant n! \cdot C_{max_1} \cdot C_{max_2} \cdots C_{max_n} \cdot p_{max_1} \cdot p_{max_2} \cdots p_{max_n}.$$

Следовательно,

$$maxCoeff(det(A)) \leqslant n! \cdot C_{max_1} \cdots C_{max_n} \cdot p_{max_2} \cdots p_{max_n}.$$

Используя формулу Стирлинга, получаем:

$$maxCoeff(det(A)) \leqslant \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{1/(12n)} \cdot C_{max_2} \cdots C_{max_n} \cdot p_{max_1} \cdot p_{max_2} \cdots p_{max_n}. \tag{3}$$

Так как каждый элемент присоединенной матрицы является алгебраическим дополнением некоторого элемента исходной матрицы, то для него можно использовать оценку (3).

Для оценки максимальной степени полинома, который является элементом присоединенной матрицы, можно взять сумму максимальных степеней полиномов по каждой строке или по каждому столбцу.

3 Параллельный алгоритм вычисления присоединённой матрицы

Пусть имеется кластер с n процессорами с номерами $0,\ 1,\ 2, \ldots,\ (n-1)$. Исходная матрица A с элементами из кольца полиномов многих переменных $Z[x_0, x_1, \ldots, x_l]$ находится на нулевом процессоре.

Параллельный алгоритм вычисления присоединённой матрицы состоит из четырех этапов.

Этап 1. Определение множества модулей.

Матрица A посылается всем процессорам. Каждый процессор, получив матрицу, находит верхнюю оценку наибольшего по абсолютной величине числового коэффициента и степень полинома для элементов присоединённой матрицы по каждой переменной. По найденным оценкам вычисляется необходимое для данной задачи число числовых и полиномиальных модулей по каждой переменной. Это дает возможность найти число r — количество простых чисел p, которые нужно взять для задания конечных полей Z/pZ, в которых будут вычисляться присоединённые матрицы.

Этап 2. Решение задачи в гомоморфных образах в конечном поле.

Желательно раздать на каждый из n процессоров одинаковое число модулей. Если n/r – дробное число, то обозначим через m остаток от деления r на n. Каждый из процессоров с номерами $0,\ 1,\ 2, \ldots,\ m-1$ выполняет вычисление присоединённой матрицы в конечном поле в $\left\lfloor \frac{r}{n} \right\rfloor + 1$ точке, каждый из остальных процессоров – в $\left\lfloor \frac{r}{n} \right\rfloor$ точках. В результате на каждом процессоре будут получены присоединенные матрицы в разных конечных поля.

Отметим, что для вычисления присоединённой матрицы в конечном поле используется алгоритм, который приведен в [1, 2].

Все процессоры имеют одну и ту же исходную матрицу и производят вычисления по такому количеству модулей, которое может отличаться не более, чем на 1. Таким образом задача равномерно распределяется по всем процессорам.

Этап 3. Восстановление элементов присоединённой матрицы в исходном кольце.

Каждая присоединённая матрица разбивается на n частей, которые содержат примерно одинаковое число элементов одинаковым образом. Процессоры обмениваются полученными частями так, что i-й процессор получает все i-ые части каждой матрицы. Это обеспечивает равномерную нагрузку на процессоры на этом этапе.

Каждый процессор восстанавливает одну часть искомой присоединённой матрицы. Восстановление элементов присоединенной матрицы проводится с использованием решения Ньютона китайской задачи об остатках [3].

Этап 4. Сбор результата на нулевом процессоре.

После того, как получены искомые элементы присоединенной матрицы, каждый процессор посылает результат нулевому процессору. Нулевой процессор объединяет их в искомую присоединённую матрицу.

4 Заключение

Предложен алгоритм вычисления присоединённой матрицы в кольце полиномов многих переменных. Алгоритм был программно реализован в системе компьютерной алгебры Mathpar[6]-[8]. Важной особенностью алгоритма является равномерная нагрузку на узлы кластера, что обеспечивает хорошую масштабируемость. Если число процессоров в кластере позволяет, то можно дополнительно распараллелить и вычисление присоединенной матрица по одному модулю. Это будет эффективно только для матриц, у которых больше, чем 256×256 элементов, см. [4, 5].

ЛИТЕРАТУРА

1. *Малашонок Г.И.* Матричные методы вычислений в коммутативных кольцах. Монография. Тамбов: Изд-во ТГУ им. Г.Р. Державина, 2002. С. 78-82.

2. *Малашонок Г.И.* О вычислении ядра оператора, действующего в модуле. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 13. Вып. 1. 2008. С. 129-131.

3. *Бухбергер Б., Калме Ж., Калтофен Э. и др.* Компьютерная алгебра: Символьные и алгебраические вычисления: Пер. с англ./Под ред. Б. Бухбергера, Дж. Коллинза, Р. Лооса. М.: Мир, 1986.

4. *Бетин А.А.* Параллельный рекурсивный алгоритм вычисления присоединенной матрицы. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 1. 2009. С. 265-269.

5. *Бетин А.А.* Эксперименты с параллельным алгоритмом вычисления присоединённой матрицы и параллельным умножением файловых матриц. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 15. Вып. 1. 2010. С. 341-345.

6. *Malaschonok G.I.* Project of Parallel Computer Algebra // Tambov University Reports. Series: Natural and Technical Sciences. V. 15. Issue 6. 2010. P. 1724-1729.

7. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 1. 2010. С. 322-327.

8. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 4. 2009. С. 744-748.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

A PARALLEL ALGORITHM FOR MODULAR CALCULATION OF AN ADJOINT MATRIX FROM POLYNOMIAL RING OF SEVERAL VARIABLES

© **Andrey Andreevich Betin**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, assistance lecturer of Mathematical Analysis Department, e-mail: andrey.betin@gmail.com

Key words: calculation of an adjoint matrix, parallel algorithm, system Mathpar.

We present a parallel modular algorithm for calculation of an adjoint matrix from polynomials ring of several variables, calculated an upper bound on the maximum absolute value of the numerical coefficient of the adjoint matrix elements. We discuss the implementation of algorithms in the system Mathpar.

УДК 519.688

ПАРАЛЛЕЛЬНЫЙ МОДУЛЯРНЫЙ АЛГОРИТМ ВЫЧИСЛЕНИЯ ХАРАКТЕРИСТИЧЕСКОГО ПОЛИНОМА МАТРИЦЫ В КОЛЬЦЕ ПОЛИНОМОВ МНОГИХ ПЕРЕМЕННЫХ

© **О. Н. Переславцева**

Ключевые слова: вычисление характеристического полинома, параллельный алгоритм, система Mathpar.

Приводится параллельный модулярный алгоритм вычисления характеристического полинома матрицы в кольце полиномов многих переменных. Обсуждается реализация в системе Mathpar.

Параллельная реализация алгоритмов позволяет решать задачи для матриц над полиномами сверхбольших размеров, которые были недоступны для однопроцессорных компьютеров. На первый план при этом выходят проблемы распараллеливания алгоритма решения матричной задачи. В компьютерной алгебре для параллельных вычислительных систем часто применяется модулярное распараллеливание. Этот тип распараллеливания появился как результат применения модулярной арифметики. Его преимущество состоит в том, что операции, связанные с разными модулями, могут выполняться одновременно.

Пусть n — количество процессоров кластера с номерами 0, 1, $2, \ldots,$ $(n-1)$ и на нулевом процессоре имеется исходная матрица A над полиномами из кольца $Z[x_1, \ldots, x_t]$. Пусть множество M, которое содержит необходимый список простых чисел, имеется на каждом процессоре.

Параллельный алгоритм вычисления характеристического полинома матрицы состоит из четырех этапов.

Этап 1. Вычисление количества модулей.

Нулевой процессор находит верхнюю оценку наибольшего по абсолютной величине числового коэффициента элементов характеристического полинома матрицы и верхние оценки наибольших степеней элементов характеристического полинома матрицы по каждой переменной. Можно использовать оценки, полученные в работе [1]. По найденным оценкам и множеству числовых модулей M вычисляется необходимое для данной задачи число числовых и полиномиальных по каждой переменной модулей. В качестве полиномиальных модулей по переменной x_i берём полиномы $x_i,\ x_i - 1,\ x_i - 2, \ldots,\ i = 1, \ldots, t$.

Затем производится рассылка исходной матрицы и количества модулей на все процессоры.

Этап 2. Решение задачи в гомоморфных образах в конечном поле.

Каждый процессор, получив матрицу и количество модулей, вычисляет число r — количество точек, в которых необходимо решить задачу в гомоморфных образах в конечном поле.

Обозначим через m остаток от деления r на n. Каждый из процессоров с номерами $0,\ 1,\ 2, \ldots,\ m-1$ выполняет вычисление характеристического полинома матрицы в конечном поле в $\left\lfloor \frac{r}{n} \right\rfloor + 1$ точке, каждый из остальных процессоров в $\left\lfloor \frac{r}{n} \right\rfloor$ точках. В результате на каждом процессоре имеется массив полиномов с элементами их конечного поля.

Для вычисления характеристического полинома матрицы в конечном поле используется алгоритм из работ [2-4].

Все процессоры имеют одну и ту же исходную матрицу и производят вычисления по одинаковому количеству модулей, количество может отличаться на один модуль, таким образом, задача равномерно распределяется по всем процессорам.

Этап 3. Восстановление элементов характеристического полинома матрицы в исходное кольцо.

На всех процессорах каждый из полученных полиномов разбивается на n частей, которые содержат приблизительно одинаковое количество мономов. Причём, j-я часть одного полинома содержит мономы тех же степеней, что и j-я часть другого полинома, $j = 0, \ldots, n-1$. Для $j \neq i$ мономы, содержащиеся в j-й части любого полинома, имеют степени переменных, отличные от какого-либо монома, содержащегося в i-й части.

Процессоры обмениваются полученными частями полиномов так, что i-й процессор получает все i-е части полиномов, $i = 0, \ldots, n-1$. На каждом процессоре будут вычисляться либо r коэффициентов, либо $r+1$. Тем самым будет обеспечена равномерная нагрузка на процессорах.

Восстановление элементов характеристического полинома матрицы проводится с использованием алгоритма Ньютона [5].

Этап 4. Сбор результата на нулевом процессоре.

После восстановления каждый процессор посылает свой результат нулевому процессору. Затем нулевой процессор объединяет полученные данные в искомый характеристический полином матрицы.

Предложенный алгоритм вычисления характеристического полинома матрицы в кольце полиномов многих переменных был реализован в системе компьютерной алгебры Mathpar [6]-[8].

Отметим, что рассмотренный алгоритм даёт равномерную нагрузку на узлы кластера и обеспечивает хорошую масштабируемость.

ЛИТЕРАТУРА

1. *Переславцева О.Н.* Вычисление характеристического полинома для полиномиальных матриц // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 1. 2009. С. 274-277.

2. *Переславцева О.Н.* О вычислении характеристического полинома матрицы // Дискретная математика. Т. 23. Вып. 1. 2011. С. 28-45.

3. *Pereslavtseva O. N.* Calculation of the characteristic polynomial of a matrix // Discrete Mathematics and Applications. Volume 21. Issue 1. Pages 109–129.

4. *Переславцева О.Н.* О вычислении коэффициентов характеристического полинома. Вычислительные методы и программирование: новые вычислительные технологии. 2008. Т. 9. № 1. С. 366-370.

5. *Бухбергер Б., Калме Ж., Калтофен Э. и др.* Компьютерная алгебра: Символьные и алгебраические вычисления: Пер. с англ./Под ред. Б. Бухбергера, Дж. Коллинза, Р. Лооса. М.: Мир, 1986.

6. *Malaschonok G.I.* Project of Parallel Computer Algebra // Tambov University Reports. Series: Natural and Technical Sciences. V. 15. Issue 6. 2010. P. 1724-1729.

7. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 1. 2010. С. 322-327.

8. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 4. 2009. С. 744-748.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

PARALLEL ALGORITHM FOR COMPUTING OF CHARACTERISTIC POLYNOMIALS OF POLYNOMIAL MATRICES

© **Oxana Nikolaevna Pereslavtseva**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, assistance lecturer of Mathematical Analysis Department, e-mail: oxana.pereslavtseva@gmail.com

Key words: computing characteristic polynomials, parallel algorithm, system Mathpar.

There is produced a parallel algorithm for computing characteristic polynomials for polynomial matrices. The algorithm is based on the method of homomorphic images in the ring of integers and in the ring of polynomials. We discuss the implementation of algorithms in the system Mathpar.

УДК 519.688

ОБ АЛГОРИТМЕ ФАКТОРИЗАЦИИ ПОЛИНОМОВ МНОГИХ ПЕРЕМЕННЫХ

© **Д. С. Ивашов**

Ключевые слова: факторизация полиномов, алгоритм освобождения от квадратов, система компьютерной алгебры Mathpar.

Предлагается последовательный алгоритм факторизации полиномов многих переменных, который входит в состав библиотеки алгоритмов системы компьютерной алгебры Mathpar.

1 Введение

Задача факторизации полиномов возникает во многих областях символьных вычислений. Исследования в области факторизации в основном сосредоточены над разложением полиномов на множители над полем целых и рациональных чисел. Разложение на множители играет важную роль в качестве подзадачи для многих других проблем, включая упрощение, символьные интеграции и решения полиномиальных уравнений. Полиномиальные разложения необходимы в таких областях, как алгебраическая теория кодирования, криптография, теория чисел.

До конца шестидесятых годов прошлого века факторизацию полиномов было принято считать «трудной» задачей, потому что не было быстрых алгоритмов для решения этой проблемы. Кроме того, были не ясны параллели между факторизацией полиномов и факторизацией целых чисел, что обе эти проблемы могут быть решены за полиномиальное время. В 1969 году Берлекэмп предложил первый алгоритм для факторизации полиномов над конечными полями. Время, требуемое для выполнения предложенного алгоритма, определялось как полиномиальная функция.

В настоящее время получены новые более быстрые алгоритмы факторизации полиномов. Марк Ван Хойя, основываясь на работах Калтофена и Шоупа, предложил систему оценок, следуя которой выбирается определенная стратегия факторизации полиномов. Сейчас появляются различные модификации известных алгоритмов, позволяющие ускорить процесс разложения на множители.

Настоящая работа продолжает исследования по факторизации полиномов многих переменных и предлагает программную реализацию алгоритмов факторизации в системе компьютерной алгебры Mathpar.

В параграфе 2 описан алгоритм факторизации полиномов многих переменных и приведена программная реализация алгоритма.

В параграфе 3 описан алгоритм выделения множителей, имеющих различные наборы переменных. Приведена программная реализация и найдены оценки стоимости этого алгоритма.

В параграфе 4 приведен пример решения задачи факторизации полинома многих переменных.

В параграфе 5 приведен пример факторизации полинома многих переменных в системе Mathpar.

2 Алгоритм факторизации полиномов многих переменных

Рассмотрим алгоритм факторизации полиномов многих переменных над рациональными числами. Пусть $F(x_1, \ldots, x_n)$ — полином от n переменных, который требуется разложить на неприводимые сомножители. Полином неприводим, если его нельзя представить в виде произведения полиномов с рациональными коэффициентами. Пусть $F(x_1, \ldots, x_n)$ — примитивен, т.е. его коэффициенты — целые числа, наибольший общий делитель (НОД) которых равен 1. Если F не является примитивным, то разделим его на НОД коэффициентов.

Алгоритм факторизации полиномов многих переменных состоит из четырех этапов.

Этап I. Разложение полиномов на множители, имеющие различные наборы переменных.

Пусть $F(x_1, \ldots, x_n)$ — полином n переменных $x_1, \ldots, x_n$. У такого полинома может быть не более 2^n сомножителей, которые имеют различные наборы переменных. При этом сомножителей, имеющих p различных переменных, может быть не более $\left(\begin{array}{c} n \\ p \end{array} \right)$. Процесс представления полинома в виде произведения таких сомножителей описан в параграфе 3.

В результате $F = \prod_{k=1}^{m} h_k$.

В системе компьютерной алгебры *Mathpar* имеется процедура *toFactorsOfCoeffs()*, реализующая данный алгоритм.

Этап II. Выделение сомножителей имеющих различную кратность.

Найдём множители, которые входят в полином в разных степенях:

$$h_k = \prod_{j=1}^{r_k} g_{kj}^{s_{kj}},$$

где $s_{ki} \neq s_{kj}$ при $i \neq j$. Множители g_{kj} имеют такой же набор переменных как и полином h_k. Процесс отделения таких сомножителей описан в [1]. В системе *Mathpar* имеется процедура *toFactorsWithDiffExps()*, реализующая данный алгоритм. При этом, если g_{kj} имеет сомножители, то каждый из них входит в g_{kj} в первой степени.

Этап III. Получение унитарного полинома.

Если старший коэффициент при старшей переменной равен 1, то это значительно облегчет дальнейшие поиски сомножителей. Процесс получения унитарного полинома подробно рассмотрен в [1]. В системе *Mathpar* имеется процедура *UnitaryPolynom()*, реализующая данный алгоритм.

Этап IV. Выделение множителей кратности единица с одинаковым набором переменных.

После первых двух этапов мы получим полиномы g_{kj}, которые не имеют кратных сомножителей и которые могут раскладываться только на взаимно простые множители. Для каждого из этих полиномов найдем множители в разложении $g_{kj} = \prod_{i=1}^{p_{kj}} f_{i_{kj}}$. На этом этапе найдем взаимно простые сомножители, используя алгоритм описанный в [1].

В системе *Mathpar* имеется процедура *toFactorsFromDiffExps*(), реализующая данный алгоритм.

В итоге получим

$$F(x_1, \ldots, x_n) = \prod_{k=1}^{m} h_k = \prod_{k=1}^{m} (\prod_{j=1}^{r_k} g_{kj}^{s_{kj}}) = \prod_{k=1}^{m} (\prod_{j=1}^{r_k} (\prod_{i=1}^{p_{kj}} f_{i_{kj}})^{s_{kj}}).$$

В системе *Mathpar* имеется процедура *factorizationInZ()*, реализующая данный алгоритм.

В данном алгоритме используется функция **length**(a). Она возвращает количество элементов массива a.

Алгоритм

Algorithm factorizationInZ (f, R)
Input: $f \in R, R = \mathbb{Z}[x_1, \ldots, x_n]$
Output: $result[\,] = f_i, result\_powers[\,] = s_i$
$a[\,] := toFactorsOfCoeffs(f)$;
for $i = 1$ to length(a) do {
    $b[i] := toFactorsWithDiffExps(a[i])$;
}
Comment: массив b — это объект, который состоит из двух массивов: $b[i].multin[\,]$ — массива множителей и $b[i].powers[\,]$ — массива их кратностей
$k := 1$;
for $i = 1$ to length(b) do {
    for $j = 1$ to length(b[i].multin) {
    $g := toFactorsFromDiffExps(b[i].multin[j])$
    for $u = 0$ to $length(g)$ do{
        $result[k] := g[u]$;
        $result\_powers[k] := b[i].powers[j]$;
        $k := k + 1$;
    }
    }
}
return $result[\,], result\_powers[\,]$;

3 Разложение полиномов на множители, имеющие различные наборы переменных

Пусть $F(x_1, \ldots, x_n)$ — полином от n переменных $x_1, \ldots, x_n$. У такого полинома может быть не более 2^n сомножителей, которые имеют различные наборы переменных. При этом сомножителей, имеющих p различных переменных из n, может быть не более $\left(\begin{array}{c} n \\ p \end{array} \right)$.

Процедура выделения этих сомножителей состоит в следующем.

Предварительно можно найти числовой сомножитель. Для этого находится НОД всех числовых коэффициентов полинома. Если НОД отличен от единицы, то поделим каждый числовой коэффициент полинома на НОД.

Затем отделяется сомножитель, не содержащий переменную x_1. Для этого нужно записать полином F по степеням переменной x_1, т.е. представить в виде:

$$F(x_1, \ldots, x_n) = \sum_{i=0}^{k} f_i(x_2, \ldots, x_n) x_1^i,$$

где k — старшая степень переменной x_1, входящей в полином $F(x_1, \ldots, x_n)$. Найдем НОД полиномиальных коэффициентов, т.е. НОД($f_1(x_2, \ldots, x_n), \ldots, f_k(x_2, \ldots, x_n)$). Этот НОД не содержит переменную x_1 и будет являться сомножителем полинома $F(x_1, \ldots, x_n)$. После деления исходного полинома $F(x_1, \ldots, x_n)$ на этот НОД, получим второй сомножитель, который будет содержать переменную x_1. Таким образом получим два сомножителя исходного полинома $F(x_1, \ldots, x_n)$, один из которых не содержит переменную x_1.

Для каждого из получившихся полиномов будем продолжать этот процесс по переменным $x_2, x_3, \ldots, x_s$. На этом этапе мы можем получить не более 2^s сомножителей. В каждом из этих полиномов попробуем выделить сомножитель, не содержащий переменную x_{s+1}. Для этого в каждом из этих полиномов найдем НОД полиномиальных коэффициентов при неизвестной x_{s+1} и разделим на него. Общее число полиномиальных сомножителей может при этом, в худшем случае, удвоиться.

Таким образом, завершив этот процесс на последней переменной x_n, получим, в худшем случае, 2^n сомножителей.

С помощью данного алгоритма мы выделим сомножители f_i, имеющие различные наборы переменных, полинома $f(x_1, \ldots, x_n)$.

В данном алгоритме используются следующие функции.

Polynomial_GCD(f,i,R) возвращает полиномиальный НОД полинома f относительно i-ой переменной из кольца R.

isNotOne(f) возвращает $true$, если полином f отличен от единицы, иначе — $false$.

Рассмотрим алгоритм разложения полиномов на множители, имеющие различные наборы переменных.

Алгоритм

Algorithm toFactorsOfCoeffs (f, R)
Input: $f \in R, R = \mathbb{Z}[x_1, \ldots, x_n]$
Output: $result[\,] = f_i$
h — количество переменных

```
result[0]:=f;
n:=1;
number:=1;
for   i := 0  to  h  do
     for   j := 0  to  n  do{
       first:=Polinomial_GCD(result[ j ],i,R);
       if(isNotOne(first)){
         rezult[number]:=divide(rezalt[ j ], first);
         rezult[j]:=first;
         number++;}
       }
     n:=number
return result;
```

Определим количество операций данного алгоритма. Пусть

G — операция вычисления НОД двух полиномов,

D — операция деления двух полиномов,

n — количество переменных исходного полинома,

m — количество мономов исходного полинома.

Для вычисления одного шага редукции требуется $(m-1)G+D$ операций. Тогда для того, чтобы разложить полином многих переменных на сомножители, имеющие разные наборы переменных, потребуется не более $(2^n-2)((m-1)G+D)$ операций.

4 Пример задачи факторизации полинома многих переменных

Постановка задачи.

Рассмотрим произведение полиномов $F(x,y,z)=(x-y)(x+y)(x+3y^2)^2(x-yz)(xz+y)(y-z)^2(y+2z^2)$. После раскрытия всех скобок будет получен полином, который для своей записи на этой странице потребует 10 строчек. Требуется разложить его на множители и получить исходное выражение.

Покажем отдельные этапы решения и используемые процедуры.

1. Выделение сомножителей, имеющих различные наборы переменных.

Выделим в полиноме F сомножители с различными наборами переменных. Для этого применим процедуру $toFactorsOfCoeffs()$. В результате получим массив сомножителей полинома F:

$f_1=-9y^6+9x^2y^4-6xy^4+6x^3y^2-x^2y^2+x^4$,

$f_2=-xyz^2-y^2z+x^2z+xy$,

$f_3=2z^4-4yz^3+2y^2z^2+yz^2-2y^2z+y^3$.

2. Выделение сомножителей, имеющих различную кратность.

В каждом из полиномов f_1, f_2, f_3 выделим сомножители, имеющие различные кратности. Для этого к f_1, f_2, f_3 применим процедуру $toFactorsWithDiffExps()$, реализующую данный алгоритм. Получим

$f_{1,1}=-y^2+x^2$ кратности 1,

$f_{1,2}=x+3y^2$ кратности 2,

$f_2=-xyz^2-y^2z+x^2z+xy$ кратности 1,

$f_{3,1} = y - z$ кратности 2,

$f_{3,2} = y + 2z^2$ кратности 1.

3. Выделение сомножителей, имеющих кратность единица.

После первых двух этапов мы получили полиномы: $f_{1,1}, f_{1,2}, f_2, f_{3,1}, f_{3,2}$. Для каждого из этих полиномов найдем взаимно простые сомножители кратности 1 с помощью процедуры $toFactorsFromDiffExps()$. Получим следующие сомножители:

$f_{1,1,1} = x - y$, $f_{1,1,2} = x + y$, $f_{1,2} = x + 3y^2$, $f_{2,1} = x - yz$, $f_{2,2} = xz + y$, $f_{3,1} = y - z$, $f_{3,2} = y + 2z^2$.

Таким образом, полином F можно представить в виде: $F(x, y, z) = f_1 * f_2 * f_3 = f_{1,1} * f_{1,2}^2 * f_2 * f_{3,1}^2 * f_{3,2} = (f_{1,1,1} * f_{1,1,2})(f_{1,2})^2(f_{2,1} * f_{2,2})(f_{3,1})^2(f_{3,2}) = (x - y)(x + y)(x + 3y^2)^2(x - yz)(xz + y)(y - z)^2(y + 2z^2)$.

5 Факторизация полиномов многих переменных в системе Mathpar

Разработанные алгоритмы факторизации полиномов многих переменных над кольцом целых чисел применяются в системе компьютерной алгебры Mathpar [7], [8], [9].

Для того чтобы разложить полином многих переменных на множители в системе Mathpar необходимо:

1) задать пространство переменных (SPACE),

2) задать полином,

3) задать функцию факторизации (Factor).

Пространство переменных определяется типом числового множества и именами переменных. Пользователь может сменить пространство переменных, например, задав команду «SPACE=Z[x,y,z]».

Пример 1. Факторизовать полином $f = -y^4z^2 + x^2y^2z^2 - 2xy^3z + 2x^3yz - x^2y^2 + x^4$.

Ввод данных в системе Mathpar

SPACE=Z[x,y,z];

$f = -y^4z^2 + x^2y^2z^2 - 2xy^3z + 2x^3yz - x^2y^2 + x^4$;

h=Factor(f);

print(h);

Система Mathpar выдаст ответ: $(x + yz)^2(x - y)(x + y)$.

ЛИТЕРАТУРА

1. *Ivashov D.S.* Factorization of polynomials of several variables // Tambov University Reports. Series: Natural and Technical Sciences. V. 16. Issue 1. 2011. P. 133-137.

2. *Ivashov D.S.* An algorithm of factorization of polynomials of several variables // Tambov University Reports. Series: Natural and Technical Sciences. V. 15. Issue 1. 2010. P. 331-334.

3. *Shoup V.* A new polynomial factorization algorithm and its implementation // Journal of Symbolic Computation V. 20. 1995. P. 363-397.

4. *Shoup V., Kaltofen E.* Subquadratic-time factoring of polynomials over finite fields // 27th Annual ACM Symposium on Theory of Computing 1995. P. 398-406.

5. *Hoeij M.V.* Factoring polynomials and the knapsack problem // Journal of Number Theory. V. 95. 2002. P. 167-189.

6. *Kaltofen E.* Polynomial factorization in comuter algebra // Springer-Verlag, 1982.

7. *Malaschonok G.I.* Project of Parallel Computer Algebra // Tambov University Reports. Series: Natural and Technical Sciences. V. 15. Issue 6. 2010. P. 1724-1729.

8. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 1. 2010. С. 322-327.

9. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 4. 2009. С. 744-748.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

AN ALGORITHM OF FACTORIZATION OF MULTIVARIATE POLYNOMIALS

© **Ivashov Dmitriy Sergeyevich**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Mathematical Analysis Department, e-mail: ivashovdmitry@gmail.com

Key words: factorization of polynomials, computer algebra, Mathpar computer algebra system.

We present and discuss a sequential algorithm for factorization of polynomial of several variables , which is a part of the library of Mathpar computer algebra system.

УДК 519.688

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ, ОПЕРАТОРЫ ВЕТВЛЕНИЯ И ЦИКЛОВ В СИСТЕМЕ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

© **Е. В. Дубовицкий**

Ключевые слова: процедурное программирование, оператор ветвления, операторы циклов, система компьютерной алгебры, система компьютерной алгебры Mathpar.
Обсуждаются алгоритмы для создания системы процедурного программирования на основе системы компьютерной алгебры. Эти алгоритмы обеспечивают создание процедур и функций, а так же создание операторов ветвления и цикла. Описанный подход был применён в системе компьютерной алгебры Mathpar.

1 Введение

На данный момент существует достаточно много систем компьютерной математики, позволяющих производить численные и аналитические вычисления. Они широко используются в образовании, научных расчётах, при решении различных инженерных задач. Среди таких систем выделяется отдельный класс, ориентированный на символьные аналитические вычисления — это системы компьютерной алгебры. В данный класс входят такие системы как Mathematica, MAPLE, CoCoA, Maxima и другие.

Любое программное обеспечение для использования в алгебраических вычислениях должно включать в себя метод представления сложных математических объектов, язык, позволяющий манипулировать с ними, и библиотеку функций для выполнения необходимых алгебраических операций.

С развитием подобных систем встает вопрос об использовании в них не только поименованных операторов, которые являются некоторым представлением математической формулировки задачи, но и такие языковые конструкции как:

- операторы ветвления if-then-else;
- операторы циклов while и for;
- процедуры и функции.

Данные конструкции расширяют возможности языка, давая пользователю инструменты для управления последовательностью действий в системе компьютерной алгебры.

Исходя из свойства структурного программирования, перечисленные выше операторы позволят использовать в системе любой алгоритм действий, заданый пользователем.

2 Постановка задачи

Для добавления к системе компьютерной алгебры операторов управления необходимо решить следующие задачи.

1. Задать булеву алгебру.
2. Задать синтаксис операторов.
3. Разработать схему компиляции исходного кода программы в промежуточное представление. Имея некоторый способ хранения нечисловых данных внутри системы, достаточно выделить область для хранения всех вхождений операторов, представляющих собой композицию функций, и создать список действий, указывающий на последовательность выполнения данных операторов.

Таким образом, задача разбивается на три подзадачи. Наиболее трудоемким является решение третьей подзадачи, оно же представляет наибольший интерес, поэтому ему посвящена большая часть данной статьи.

3 Операторы ветвления и циклов

Если первые два этапа не представляют сложности и сводятся к добавлению операторов булевой алгебры и договорённости о синтаксисе будущих операторов управления последовательностью вычислений в системе компьютерной алгебры, то для реализации самих алгоритмов подпрограмм и операторов ветвления и циклов удобно иметь некоторое промежуточное представление структуры всей программы.

Обычно поставленную в математическом представлении задачу можно записать в виде совокупности операторов присвоения (например, присвоение через символ «=»), математических операторов (+, –, *, / и т.д.) и простых функций (sin, cos и т.д.).

Пример 1.
$a = cos(x) + sin(x)$;
$b = 2 * sin(x)$;
$a = a - b$;
$b = a/cos(x)$;
$c = a * b$;

В итоге мы получаем несколько поименованных выражений, представляющих собой композицию различных функций.

В системе компьютерной алгебры Mathpar ([1], [2]) существует тип Fname для хранения поименованных выражений. Структура объекта типа Fname приведена на рис. 1.

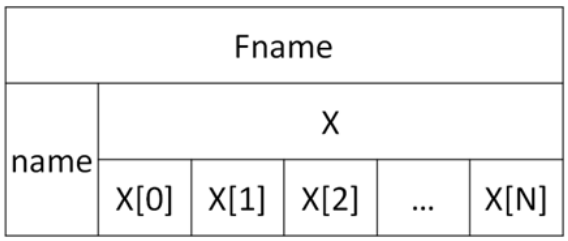

Рис. 1. Структура типа Fname

Fname имеет два поля:

- name — имя функции, которое представляет собой поле строкового типа;
- X — массив представлений функции, встретившихся в исходном коде пользовательской программы, с именем name. Это поле является массивом типа Element.

Тип Element представляет собой способ хранения числовых и нечисловых данных в системе компьютерной алгебры Mathpar ([1], [2]). Имея массив или список элементов типа Fname, сохраняем в памяти все встретившиеся в программе выражения на момент выполнения программы. Пример способа хранения данных программы приведён на рис. 2.

public java.util.ArrayList<Element> **funcs**				
a	**X**			
	X[0]	**X[1]**	**...**	**X[N]**
	cos(x)+sin(x)	a-b	...	
b	**X**			
	X[0]	**X[1]**	**...**	**X[N]**
	2*sin(x)	a/cos(x)	...	
c	**X**			
	X[0]	**X[1]**	**...**	**X[N]**
	a*b		...	

Рис. 2. Способ хранения данных программы

После получения такой структуры представляем исходный код программы в виде ссылок на элементы данного списка. Таким образом, исходная программа сводится к внутреннему представлению, которое содержит в себе список типа Fname и поле со ссылками на этот список.

Например, программа с исходным кодом из примера 1 и списком, представленным на рис. 2, примет вид: 00 10 01 11 20.

Сохранение выражений и создание списка последовательности действий выполняются в процессе трансляции исходного кода программы. После завершения трансляции исходный код программы далее не требуется, поэтому код заменяется внутренним представлением программы.

Таким образом, мы получили удобный способ представления исходного кода, который позволит выполнять программу любое количество раз без повторной трансляции. Подобное представление позволяет добавить операторы ветвления и циклов в систему компьютерной алгебры.

Опишем синтаксис операторов.

- оператор ветвления: `if (<условие>) {<ветка_1>} else {<ветка_2>}`;
- оператор цикла с предусловием: `while (<условие>) {<тело_цикла>}`;
- оператор цикла со счётчиком:

`for (<инициализация_счётчика>; <условие_остановки_цикла>; <инкремент_счётчика>) {<тело_цикла>}`.

В список последовательности действий необходимо добавить операторы безусловного и условного переходов.

- безусловный переход: `-1 <индекс_перехода>`. Например, -1 6;
- условный переход: `-2 <ячейка_условия><индекс_перехода>`. Например, -2 4 0 16.

После введённых обозначений задача сводится к верному распознаванию операторов в тексте программы и к правильному построению списка действий.

Рассмотрим список действий для каждого из операторов управления на примерах.

- Оператор ветвления

  if (a < b) {a = a + b;}

  Список действий: -2 0 0 6 1 0

  if (a < b) {a = a + b;} else {a = a + b +5;}

  Список действий: -2 0 0 8 1 0 -1 10 1 1

- Оператор цикла с предусловием

  while (a < b) {a = a + 1;}

  Список действий: -2 0 0 8 1 0 -1 0

- Оператор цикла со счётчиком

  for (i = 0; i < 5; i = i + 1) {c = c + a;}

  Список действий: 0 0 -2 1 0 12 2 0 0 1 -1 2

В цикле со счётчиком место до оператора условного перехода занимает ячейка, отвечающая за инициализацию счётчика, а место до оператора безусловного перехода — ячейка, отвечающая за изменение счётчика.

4 Процедуры и функции

Для добавления к системе компьютерной алгебры процедур и функций записываем процедуру или функцию в отдельный список действий и при вызове исполняем его по определённым правилам. Запись списка действий осуществляется в список или массив, который впоследствии записывается в `ArrayList` наряду с основной программой.

Отличием функции от процедуры является наличие возвращаемой величины, а также возможность использования функции в качестве операнда в выражениях. Таким образом, можно не различать синтаксически эти два оператора и остановиться на обозначении процедуры, но со следующим условием — любая процедура возвращает значение, либо записанное пользователем после оператора `return`, либо пустое значение `null`. Поэтому синтаксис для процедур имеет следующий вид.

- Описание процедуры и функции:

```
procedure <имя_процедуры> (<аргументы_(через_запятую)>) {
    тело_процедуры
}
```

- Вызов процедуры: `<имя_процедуры>(передаваемые_аргументы_(через_запятую))`.

При отсутствии аргументов ставятся пустые скобки.

Для оператора `return` в списке действий также введем постоянную со значением -3. Она обозначит выход из метода, выполняющего процедуру, с возвращаемым значением, взятым из ячейки, записанной после данной постоянной.

При вычислении выражения, в составе которого есть процедура, сначала вычисляем процедуру, то есть получаем числовое или символьное представление решения процедуры, затем подставляем это значение в выражение.

5 Заключение

Таким образом, в системе компьютерной алгебры Mathpar появились возможности использования нелинейных конструкций кода, а также возможности структурного программирования. Это позволило упорядочить код и дало пользователю некоторые дополнительные инструменты для решения задач.

ЛИТЕРАТУРА

1. *Malaschonok G.I.* Project of Parallel Computer Algebra. Tambov University Reports. Series: Natural and Technical Sciences. V. 15. Issue 6. 2010. P. 1724-1729.

2. *Малашонок Г.И.* Компьютерная математика для вычислительной сети. Вестник Тамбовского университета. Сер.: Естественные и технические науки. Т. 15. Вып. 1. 2010. С. 322-327

3. *Дэвенпорт Дж, Сирэ И., Турнье Э.* Компьютерная алгебра, Мир, 1991.

4. Мир ПК [Электронный ресурс]. / ЗАО «Издательство «Открытые системы»». Электрон. журн. Режим доступа к журн.: http://www.osp.ru/pcworld/. Загл. с экрана.

5. *Ахо Альфред В., Лам Моника С., Сети Рави, Ульман Джеффри Д.* Компиляторы: принципы, технологии и инструментарий, 2 издание — М.: «Вильямс», 2010.

6. *Хантер Р.* Основные концепции компиляторов — М.: Вильямс, 2002.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

PROCEDURAL PROGRAMMING, OPERATORS OF BRANCHING AND LOOPING IN THE COMPUTER ALGEBRA SYSTEM

© **Dubovitsky Evgeny Vladimirovich**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Mathematical Analysis Department, e-mail: dubovitskyevgeny@gmail.com

Key words: procedural programming, operators of branching and looping, computer algebra system Mathpar.

We discuss algorithms for creating a system of procedural programming, based on computer algebra system. These algorithms ensure the creation of procedures and functions, as well as the establishment of branch and loop statements. The described approach was applied in the computer algebra system Mathpar.

УДК 519.688

ПРЕОБРАЗОВАНИЕ КОМПОЗИЦИЙ ФУНКЦИЙ, СОДЕРЖАЩИХ ТРИГОНОМЕТРИЧЕСКИЕ ФУНКЦИИ

© Д. И. Шляпин

Ключевые слова: тригонометрические функции, композиция функций, компьютерная алгебра.

Приводятся алгоритмы преобразования композиции функций, которые содержат тригонометрические функции. Алгоритмы основаны на тригонометрических тождествах и дробно-рациональных преобразованиях. Алгоритмы используются в системе компьютерной алгебры Mathpar.

1 Введение

Одной из важных задач компьютерной алгебры является задача преобразования композиций трансцендентных функций к определённым видам. Композицию функций можно рассматривать как функциональное дерево. Преобразование функции сводится к некоторой стратегии обхода такого дерева и преобразования его поддеревьев с помощью функциональных тождеств. Будем рассматривать следующие два способа преобразований.

1. Преобразование стоящее в тождественной замене суммы или произведения композиций трансцендентных функций одной трансцендентной функцией или числом. Будем называть такой способ **симплификацией**.

2. Преобразование, обратное симплификации будем называть **разложением**.

В этой работе мы рассмотрим преобразования функций, содержащих тригонометрические функции. Для симплификации будем использовать следующие тождества.

$$1 = \sin^2(\alpha) + \cos^2(\alpha), \tag{1}$$

$$\cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \mp \sin(\alpha)\sin(\beta), \tag{2}$$

$$\cos(2\alpha) = \cos^2(\alpha) - \sin^2(\alpha), \tag{3}$$

$$\sin(\alpha \pm \beta) = \sin(\alpha)\cos(\beta) \pm \cos(\alpha)\sin(\beta), \tag{4}$$

$$\sin(2\alpha) = 2\sin(\alpha)\cos(\alpha). \tag{5}$$

2 Переход от композиции трансцендентных функций к рациональной функции многих переменных

Пусть дана функция $F(x_1, \ldots, x_n)$, которая является композицией трансцендентных функций. «Листьями» дерева этой функции являются дробно-рациональные функции переменных $x_1, \ldots, x_n$.

Выделим все различные трансцендентные функции, которые расположены на верхнем уровне дерева данной функции. Каждая функция, входящая в композицию, является поддеревом дерева данной функции. Заменим каждую из функций новой переменной u_i, $i = 1, \ldots, k$. В результате получим дробно-рациональную функцию $P(x_1, \ldots, x_n, u_1, \ldots, u_k)$ от переменных $x_1, \ldots, x_n, u_1, \ldots, u_k$. Преобразуем функцию P к отношению двух полиномов. Найдем наибольший общий делитель числителя и знаменателя, сократим дробь на него. Сохраним его, чтобы учесть в области определения данной функции. После сокращения получим функцию $P = \frac{N}{D}$, где N и D — полиномы.

3 Алгоритм симплификации

Пусть дана композиция функций F.

1. Перейдём от функционального дерева к рациональной функции многих переменных $P = \frac{N}{D}$, где N и D — полиномы.

2. Будем упрощать отдельно многочлены N и D.

   Представим многочлены N и D в виде $N = N_1 + N_2$ и $D = D_1 + D_2$, где N_1 и D_1 содержат одночлены, в которых хотя бы одна переменная является тригонометрической функцией. Полиномы N_2 и D_2 не содержат переменных, соответствующих тригонометрическим функциям. Далее преобразовывать будем только N_1 и D_1.

3. Рассматриваем всевозможные полиномы, образованные парами мономов полинома N_1. Каждый из них раскладываем на полиномиальные множители. Если сомножитель содержит более двух мономов, то применяем рекурсивно алгоритм симплификации. Если сомножитель содержит два монома, то пытаемся найти соответствующее тригонометрическое тождество и применить его.

   Если можно выполнить симплификацию (1) — (4), то выполняем её и получаем вместо пары один новый моном.

4. Возвращаемся к шагу 3 и повторяем, пока есть хотя бы одно преобразование.

5. Проверяем возможность преобразования по формуле (5) всех мономов полученного многочлена. Если есть хотя бы одно преобразование, то выполняем преобразование и возвращаемся к шагу 3.

6. Шаги 3 — 5 выполняем для многочлена D_1.

7. Вводим обратную замену.

4 Алгоритм разложения.

Пусть дана функция $F = \prod_{i=1}^{n} \sin(\alpha_i)\cos(\beta_i)$. Аргументы α_i и β_i являются композициями трансцендентных функций, $i = 1, \ldots, n$. Если аргументы α_i и β_i являются суммами, разностями или делятся нацело на два, то к ним применяются тождества (2) — (5).

1. Аргументы α_i и β_i представим в виде $\alpha_i = a + b$ и $\beta_i = c + d$, где a и c — первые мономы соответствующего аргумента, b и d — остальные часть аргументов. Применим соответствующие тригонометрические тождества к $\sin(a+b)$ и $\cos(c+d)$.

2. Будем повторять шаг 1 до тех пор, пока применимы тождества (2) — (5).

Выполняем обратную замену.

5 Примеры

Пример 1. Упростить функцию

$$\sin(y)\cos(x)\cos(z) + \ln(x) + \mathrm{tg}(y) + \cos(y)\sin(x)\cos(z) + \sin(z)\cos(y+x) + x^3.$$

В результате выполнения алгоритма факторизации, получим

$$x^3 + \sin(z+y+x) + \ln(x) + \mathrm{tg}(y).$$

Пример 2. Представить в виде композиции функцию $\cos(x^3 + x^2 + x)$.
В результате применения к данной функции алгоритма разложения, получим

$$\cos(x^3)(\cos(x^2)\cos(x) - \sin(x^2)\sin(x)) - \sin(x^3)(\sin(x^2)\cos(x) + \cos(x^2)\sin(x)).$$

Пример 3. Представить в виде композиции функцию $\cos(\cos(x) + \sin(x))$.
В результате применения к данной функции алгоритма разложения, получим

$$\cos(\sin(x))\cos(\cos(x)) - \sin(\sin(x))\sin(\cos(x)).$$

Пример 4. Представить в виде композиции функцию $\cos(\cos(x+1) + \sin(x))$.
В результате применения к данной функции алгоритма разложения, получим

$$(\cos(\sin(x)\sin(1))(\cos(\sin(x))\cos(\cos(x)\cos(1)) - \sin(\sin(x))\sin(\cos(x)\cos(1))) -$$

$$(-1)\sin(\sin(x)\sin(1))(\sin(\sin(x))\cos(\cos(x)\cos(1)) + \cos(\sin(x))\sin(\cos(x)\cos(1)))).$$

ЛИТЕРАТУРА

1. *Хабибулин И.* Самоучитель Java 2. СПб.: БХВ-Петербург, 2005.
2. *Ноутон П., Шилдт Г.* Java 2. СПб.: БХВ-Петербург, 2008 г.

3. *Малашонок Г.И.* О проекте параллельной компьютерной алгебры // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 4. 2009. С. 744-748.

4. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 1. 2010. С. 322-327.

5. *Малашонок Г.И.* О вычислении ядра оператора, действующего в модуле // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 13. Вып. 1. 2008. С. 129-131.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

TRANSFORMATION OF COMPOSITIONS OF FUNCTIONS CONTAINING TRIGONOMETRIC FUNCTIONS

© **Dmitry Igorevich Shlapin**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Mathematical Analysis Department, e-mail: shlapin.dmitry@gmail.com

Key words: trigonometric functions, function composition, computer algebra Mathpar.

Discusses the algorithms for converting the compositions of functions, which contain trigonometric functions. We use the basic trigonometric identities and rational-fractional transformations. These algorithms are used in computer algebra system Mathpar.

УДК 519.688

К ПРОБЛЕМЕ ПОСТРОЕНИЯ АЛГОРИТМА СИМВОЛЬНОГО ИНТЕГРИРОВАНИЯ

© **С. М. Тарарова**

Ключевые слова: принцип Лиувилля об элементарном интеграле, дифференциальное поле, символьное интегрирование, полиномиальная часть интеграла, элементарные функции.

Работа посвящена интегрированию композиций элементарных функций. Интеграл представляется в виде суммы полиномиальной части и дробно-рациональной части. Трансцендентные функции, которые входят в эти части, — это логарифмические или экспоненциальные функции над дифференциальным полем. Мы рассматриваем интеграл только для полиномиальной части. В решении этой задачи центральную роль играет теорема Лиувилля. Эти алгоритмы используются в системе компьютерной алгебры Mathpar.

1 Введение

К классу элементарных функций относятся алгебраические функции, экспонента, логарифмическая функция и любые их конечные композиции. Такие композиции в комплексной области можно свести к тригонометрическим, гиперболическим и обратным им функциям.

С рациональными функциями принципиальных трудностей нет. Интеграл от рациональной функции всегда существует в виде элементарной функции. Алгоритм интегрирования дробно-рациональных функций был представлен Эрмитом [1] более века назад, который использует только полиномиальные операции без введения алгебраических расширений.

С алгебраическими функциями дело обстоит сложнее. Числа, являющиеся корнями полиномов с рациональными коэффициентами, называются алгебраическими числами. Они приводят нас к обобщению класса рациональных функций — алгебраическим функциям.

Алгебраическая функция y от переменной x задаётся неявным образом через полином от x и y:

$$p_n(x)y^n + \cdots + p_1(x)y + p_0(x) = 0,$$

где $p_i(x)$ — полиномы с рациональными коэффициентами, $i = 0, \ldots, n$.

Проверка, принадлежит ли подынтегральная функция данному классу, осуществляется с помощью структурной теоремы Риша [2]. Далее теорема Лиувилля [1] (1833г.) позволяет определить вид элементарного интеграла, если такой существует.

Роберт Риш [3] из Калифорнийского университета в 1969-1970 годах опубликовал алгоритм, приводящий любую элементарную функцию к виду $v_0' + \sum_{i=1}^{m} c_i * \frac{v_i'}{v_i}$ или определяющий, что такое приведение невозможно и, следовательно, интеграла как элементарной функции нет. Алгоритм Риша являлся на время своего опубликования алгоритмом интегрирования лишь чисто трансцендентных функций. Для алгебраических составляющих Риш привёл в общих чертах доказательство существования алгоритма, но чтобы сделать из него алгоритм, потребовался ещё не один десяток лет.

Первая реализация алгоритма Риша была выполнена Джоэлом Мозесом [4] в рамках знаменитого «Project MAC» в MIT в 1971 г. Программа под названием SIN интегрировала чисто трансцендентные функции.

Дэвенпорт [5] в 1981 г., основываясь на работе Риша и некоторых глубоких результатах дифференциальной алгебры и комплексного анализа, разработал алгоритм интегрирования чисто алгебраических функций и реализовал его в известной среде символьных вычислений REDUCE-2.

Программы Мозеса и Дэвенпорта служили для решения двух частных случаев одной задачи, но для общего случая — интегрирования произвольных элементарных функций — были одинаково непригодны.

Алгоритм Дэвенпорта ещё и обладал большой вычислительной сложностью, что его автор признавал и выражал надежду, что алгоритм можно упростить. Так и произошло: Барри Трагер [6] из MIT в 1984 г. внёс серьёзные улучшения в алгоритм Дэвенпорта. Обновлённый алгоритм обладал гораздо большей практической ценностью и был реализован в математических программах Axiom и Maple.

Решающий шаг к практическому решению вопроса сделал Мануэль Бронштейн [7] в 1990 г, обобщив алгоритм Трагера на произвольные элементарные функции. В 1998 г. Бронштейн написал монографию [8] по символьному интегрированию, в которой понятным языком изложил лучшие достижения в этой области, начиная с теоремы Лиувилля и заканчивая собственными результатами. Бронштейн был одним из ведущих разработчиков Axiom. Вместе с Трагером он реализовал в Axiom свой алгоритм, но лишь частично.

Целью данной статьи является разработка полного алгоритма нахождения интеграла от полиномиальной части композиции элементарных функции. Особое внимание уделяется специальным частным случаям. Мы опираемся на предыдущую работу [9].

2 Принцип Лиувилля

Основной результат интегрирования элементарных функций был впервые представлен Лиувиллем [10] в 1833 году. Принцип Лиувилля является базовым алгоритмическим подходом интегрирования элементарных функций.

Для формулировки принципа Лиувилля введем некоторые понятия.

Определение 1. *Дифференциальным полем называется поле F, в котором определена операция дифференцирования d, удовлетворяющая правилу Лейбница*

$$d(f * g) = d(f) * g + f * d(g)$$

и свойству дистрибутивности относительно сложения

$$d(f+g) = d(f) + d(g).$$

Пример 1. $R(x, e^x)$ — дифференциальное поле, которое содержит рациональные функции от e^x, чьи коэффициенты являются полиномами или рациональными функциями от x. Дифференциальное поле $R(x, e^x)$ является расширением поля $R(x)$, которое содержит полиномы или рациональные функции от x. Функция $\ln(x) \notin R(x, e^x)$ и $\ln(x+1) \notin R(x, e^x, \ln(x))$, но $\ln(x^2+x) \in R(x, e^x, \ln(x), \ln(x+1))$, так как $\ln(x^2+x) = \ln(x) + \ln(x+1)$. Дифференциальное поле $R(x, e^x, \ln(x), \ln(x+1))$ является расширением поля $R(x, e^x)$ и т.д.

О п р е д е л е н и е 2. *Полем констант K дифференциальноо поля F называется множество элементов поля, которые при дифференцировании обращаются в ноль:*

$$K = \{x : x \in F, d(x) = 0\}.$$

Т е о р е м а 1. Принцип Лиувилля. Пусть F — некоторое дифференциальное поле с полем констант K. Для $f \in F$ пусть существует $g \in G$ такое, что $g' = f$, т.е. $g = \int f$, где G — элементарное расширение F, имеющее такое же поле констант K. Тогда существуют $v_0, v_1, \ldots, v_m \in F$ и константы $c_1, c_2, \ldots, c_m \in K$ такие, что

$$f = v_0' + \sum_{i=1}^{m} c_i * \frac{v_i'}{v_i}.$$

Другими словами:

$$\int f = v_0 + \sum_{i=1}^{n} c_i \ln(v_i).$$

Д о к а з а т е л ь с т в о.

Предположим, что существуют $\theta_1, \ldots, \theta_N$ такие, что

$$G = F(\theta_1, \ldots, \theta_N),$$

где каждый элемент θ_i является либо логарифмической функцией, либо экспонентой над полем $F_{i-1} = F(\theta_1, \ldots, \theta_{i-1})$, $1 \leqslant i \leqslant N$. Пусть каждое расширение поля $F(\theta_1, \ldots, \theta_i)$, $1 \leqslant i \leqslant N$, имеет одинаковое поле констант K и существует $g \in G$, удовлетворяющее равенству $g' = f$. Доказательство основано на введении числа N элементарных расширений, входящих в G.

Если $N = 0$, то по условию теоремы существует $g \in F$, удовлетворяющее равенству $g' = f$. Тогда $m = 0$ и $f = v_0'$ и, следовательно, $v_0 = g$.

Предположим, что теорема верна для любого числа расширений меньше, чем N. Для случая N расширений мы можем записать поле $F(\theta_1, \ldots, \theta_N)$ в виде $F(\theta_1)(\theta_2, \ldots, \theta_N)$. Так как $f \in F(\theta_1)$ и $g \in F(\theta_1)(\theta_2, \ldots, \theta_N)$ удовлетворяют равенству $g' = f$, то мы можем предположить, что существуют $v_i(\theta_1) \in F(\theta_1)$ и константы $c_i \in K$, $1 \leqslant i \leqslant m$, для которых выполняется

$$f = v_0'(\theta_1) + \sum_{i=1}^{m} c_i * \frac{v_i'(\theta_1)}{v_i(\theta_1)}. \tag{1}$$

Для простоты обозначим θ_1 через θ.

Рассмотрим случай, когда функция θ трансцендентная и логарифмическая над F. Пусть $v_0(\theta) \in F(\theta)$ будет представлена в виде

$$v_0(\theta_1) = \frac{a(\theta)}{b(\theta)},$$

где $a,\ b \in F[\theta],\ GCD(a,b) = 1$. Мы можем представить $b(\theta)$ в виде

$$b(\theta) = \prod_{i=1}^{\mu} b_i^{r_i}(\theta),$$

где $b_i(\theta)$ — различные неприводимые полиномы в $F[\theta]$ и $r_i \in \mathbb{Z}, r_i > 0,\ 1 \leqslant i \leqslant \mu$. Разложим $v_0(\theta)$ на простейшие дроби:

$$v_0(\theta) = a_0(\theta) + \sum_{i=1}^{\mu} \sum_{j=1}^{r_i} \frac{a_{ij}(\theta)}{b_i^j(\theta)},$$

где $a_0, a_{ij}, b_i \in F[\theta]$ и $deg(a_{ij}) < deg(b_i)$. Из равенства (1) получаем

$$f = a_0'(\theta) + \sum_{i=1}^{\mu} \sum_{j=1}^{r_i} \left(\frac{a_{ij}'(\theta)}{b_i^j(\theta)} - \frac{j a_{ij}(\theta) b_i'(\theta)}{b_i^{j+1}(\theta)} \right) + \sum_{i=1}^{m} c_i \frac{v_i'(\theta)}{v_i(\theta)}. \tag{2}$$

Важным свойством равенства (2) является то, что его левая часть не зависит от θ.

Так как функция θ — логарифмическая функция над F, то существует функция $u \in F$ такая, что $\theta' = u'/u$. Пусть $p(\theta)$ — некоторый неприводимый полином в $F[x]$ положительной степени. Тогда $p'(\theta) \in F[\theta]$, если $\deg(p'(\theta)) < \deg(p(\theta))$ и, следовательно, $p(\theta)$ не делит $p'(\theta)x$. Если $p(\theta)$ является одним из $b_i(\theta)$ с максимальной степенью r_i, для некоторого $i \in \{1, \ldots, \mu\}$, то правая часть равенства (2) содержит ровно одно слагаемое, знаменатель которого есть $p^{r_i+1}(\theta)$. Тогда не существует другого слагаемого, с которым он в сумме даст ноль. Следовательно, этот многочлен должен находиться в левой части равенства, что противоречит тому, что f не зависит от θ. Из этого можно сделать вывод, что средний член равенства (2) (двойная сумма) должна быть равна 0. Если $p(\theta)$ является одним из $v_i(\theta)$, входящих в правую часть равенства (2), то существует ровно одно слагаемое, знаменателем которого является $p(\theta)$, снова противоречие. Отсюда делаем вывод, что не существуют в равенстве (2) слагаемых, имеющих в знаменателе элемент θ.

$$f = a_0'(\theta) + \sum_{i=1}^{m} c_i \frac{v_i'}{v_i},$$

где $a_0 \in F[\theta],\ v_i \in F$ и $c_i \in K,\ 1 \leqslant i \leqslant m$. Так как f и v_i не зависят от θ, то $a_0'(\theta)$ не должно зависеть от θ. Тогда

$$a_0(\theta) = c\theta + d \in F[\theta]$$

для некоторой константы $c \in K$ и некоторой функции $d \in F$. Следовательно,

$$f = d' + c\frac{u'}{u} + \sum_{i=1}^{m} c_i \frac{v_i'}{v_i},$$

где $d,\ u,\ v_i \in F$ и $c,\ c_i \in K,\ 1 \leqslant i \leqslant m$. Получили желаемую форму.

Аналогично и для случая, когда функция θ экспоненциальная над F.

3 Символьное интегрирование элементарных функций

Сформулируем задачу интегрирования в конечном виде. Пусть $F(x, \theta_1, \ldots, \theta_n)$ — дифференциальное поле и K — поле констант, x — переменная, для которой $x' = 1$, и для любого $i = 1, \ldots, n$ элемент θ_i является либо логарифмической, либо экспоненциальной функцией над полем $F_{n-1} = F(x, \theta_1, \ldots, \theta_{n-1})$.

Построим алгоритм, позволяющий для произвольной элементарной функции $f \in F_n$ найти элементарную функцию $g(x)$, для которой $g'(x) = f$, или доказать, что такой функции не существует. Алгоритм интегрирования имеет рекурсивный характер, то есть от задачи, сформулированной в терминах поля F_i, нужно перейти к одной или нескольким задачам над полем F_{i-1}.

Для вычисления полиномиальной части интеграла рассмотрим отдельно случаи, когда последнее расширение θ_n является логарифмической функцией и экспонентой. Обозначим для простоты θ_n через θ.

θ — логарифмическая функция.

Рассмотрим случай интегрирования полиномиальной части интеграла, где последнее расширение $\theta = \theta_n$ — логарифмическая функция над $F(x, \theta_1, \ldots, \theta_{n-1})$, то есть $\theta = \ln(u)$, где $u \in F_{n-1}$. Для подынтегрального выражения $f \in F(x, \theta_1, \ldots, \theta_n)$ полиномиальной частью является полином $p(\theta) \in F_{n-1}[\theta]$. Пусть $p(\theta) = \sum_{i=0}^{n} p_i \theta^i$, где $p_i \in F_{n-1}$. По принципу Лиувилля, если $\int p(\theta)$ берется в элементарном виде, то

$$p(\theta) = v_0'(\theta) + \sum_{i=1}^{m} c_i \frac{v_i'(\theta)}{v_i(\theta)}.$$

Пусть $v_0(\theta) = \sum_{i=0}^{n+1} B_i \theta^i$, где $B_i \in F_{n-1}$. Процесс интегрирования заключается в нахождении коэффициентов B_i, $i = 1, \ldots, n+1$, из равенства:

$$\sum_{i=0}^{n} p_i \theta^i = \left(\sum_{i=0}^{n+1} B_i \theta^i + \sum_{i=1}^{m} c_i \ln\left(v_i(\theta)\right) \right)' = \sum_{i=0}^{n+1} B_i' \theta^i + \sum_{i=1}^{n+1} i B_i \theta' \theta^{i-1} + \sum_{i=1}^{m} c_i \frac{v_i'(\theta)}{v_i(\theta)}.$$

С помощью метода неопределенных коэффициентов находим B_i, $i = 1, \ldots, n+1$.

$$\begin{cases} 0 = B_{n+1}', \\ p_n = B_n', \\ p_{n-1} = B_{n-1}' + n B_n \theta', \\ \ldots \\ p_k = B_k' + (k+1) B_{k+1} \theta', \\ \ldots \\ p_0 = B_0' + B_1 \theta' + \sum_{i=1}^{m} c_i \frac{v_i'(\theta)}{v_i(\theta)} = \bar{b}' + B_1. \end{cases} \quad \text{Тогда} \quad \begin{cases} B_{n+1} = 0, \\ B_n = \int p_n, \\ B_{n-1} = \int (p_{n-1} - n B_n \theta'), \\ \ldots \\ B_k = \int (p_k - (k+1) B_{k+1} \theta'), \\ \ldots \\ \bar{b} = \int (p_0 - B_1 \theta'). \end{cases}$$

Пример 2. Рассмотрим интеграл $\int \ln(x)$.

Введем обозначение $\theta = \ln(x)$, тогда $\theta' = \frac{1}{x}$. Элемент θ является трансцендентным над полем рациональных чисел $\mathbb{Q}$. Будем рассматривать θ как независимую переменную над полем $\mathbb{Q}$. Если интеграл является элементарным, то

$$\int \theta = B_2 \theta^2 + B_1 \theta + \bar{b}.$$

Приравниваем коэффициенты при соответствующих степенях θ:

$$\begin{cases} 0 = B_2', \\ 1 = B_1', \\ 0 = \bar{b}' + B_1\theta'. \end{cases} \qquad \begin{cases} B_2 = 0, \\ B_1 = \int 1 = x, \\ \bar{b} = \int -1 = -x. \end{cases}$$

Таким образом,

$$\int \ln(x) = x\ln(x) - x.$$

θ — экспонента.

Пусть $\theta = \theta_n$ — экспоненциальная функция над $F(x, \theta_1, \ldots, \theta_{n-1})$, то есть $\theta = e^u$, где $u \in F_{n-1}$. Полиномиальной частью подынтегрального выражения $f \in F(x, \theta_1, \ldots, \theta_n)$ является расширенный полином $p(\theta) \in F_{n-1}[\theta]$. Пусть $p(\theta) = \sum\limits_{i=-k}^{l} p_i\theta^i$, где $p_i \in F_{n-1}$, $i = -k, \ldots, l$. По принципу Лиувилля, если $\int p(\theta)$ берется в элементарном виде, тогда

$$p(\theta) = v_0'(\theta) + \sum_{i=1}^{m} c_i \frac{v_i'(\theta)}{v_i(\theta)}.$$

Пусть $v_0(\theta) = \sum\limits_{i=-k}^{l} q_i\theta^i$, где $q_i \in F_{n-1}$. Процесс интегрирования заключается в нахождении коэффициентов q_i, $i = -k, \ldots, l$, из равенства:

$$\sum_{i=-k}^{l} p_i\theta^i = \left(\sum_{i=-k}^{l} q_i\theta^i + \sum_{i=1}^{m} c_i \ln(v_i(\theta)) \right)' = \sum_{i=-k}^{l} q_i'\theta^i + \sum_{i=-k}^{l} iB_i\theta^i + \sum_{i=1}^{m} c_i \frac{v_i'(\theta)}{v_i(\theta)}.$$

Приравниваем коэффициенты при степенях θ, получаем

$$p_i = q_i' + i * u' * q_i \text{ при } i \neq 0, \tag{3}$$

$$p_0 = q_0 + \sum_{i=1}^{m} c_i \frac{v_i'(\theta)}{v_i(\theta)} = \bar{q}.$$

Для решения данного дифференциального уравнения сначала необходимо определить вид q_i, где q_i — рациональная функция. Пусть $q_i = \frac{a}{b}$, где a и $b \in F_{n-1}$. Если у q_i знаменатель не константа, то степень знаменателя при дифференцировании q_i возрастет и не может быть сокращена с другим слагаемым.

Возможны следующие случаи.

1. p_i — полином, $p_i \in F(x, \theta_1, \ldots, \theta_{n-1})$, то q_i тоже будет полиномом $q_i = a$, где $a \in F(x, \theta_1, \ldots, \theta_{n-1})$.

   - Если u' тоже является полиномом и $u' = v$, где $v \in F(x, \theta_1, \ldots, \theta_{n-1})$, то равенство (3) примет следующий вид:

   $$p_i = a' + iva. \tag{4}$$

   - Если u' — дробно-рациональная функция $u' = \frac{v}{w}$, где $v, w \in F(x, \theta_1, \ldots, \theta_{n-1})$ и $\deg(v) < \deg(w)$), то равенство (3) примет вид:

   $$p_i w = wa' + iva. \tag{5}$$

2. p_i — дробно-рациональная функция, т.е. $p_i = \frac{r}{g}$, где $r, g \in F_{n-1}$, то q_i тоже будет дробно-рациональной функцией $q_i = \frac{a}{b}$, где $a, b \in F(x, \theta_1, \ldots, \theta_{n-1})$ и $\deg(a) < \deg(b)$.

   - Если u' является полиномом $u' = v$, где $v \in F(x, \theta_1, \ldots, \theta_{n-1})$, то равенство (3) примет следующий вид:
   $$\frac{r}{g} = \frac{a'b - ab'}{b^2} + \frac{iva}{b}.$$
   Тогда
   $$b = \sqrt{g}, r = a'b - ab' + ivab = ba' + (-b' + ivb)a. \tag{6}$$
   - Если u' — дробно-рациональная функция $u' = \frac{v}{w}$, где $v, w \in F(x, \theta_1, \ldots, \theta_{n-1})$ и $\deg(v) < \deg(w)$, то равенство (3) примет вид:
   $$\frac{r}{g} = \frac{a'b - ab'}{b^2} + \frac{iva}{wb}$$
   и
   $$b = \sqrt{\frac{g}{w}}, r = w(a'b - ab') + ivab = wba' + (-wb' + ivb)a. \tag{7}$$

Обозначим в равенствах (4) – (7) левую часть через P, коэффициент при a' через $\bar{c}_1$, а коэффициент при a через $\bar{c}_2$. В результате получается равенство:

$$P = \bar{c}_1 a' + \bar{c}_2 a, \tag{8}$$

где $P, \bar{c}_1, \bar{c}_2, a \in F(x, \theta_1, \ldots, \theta_{n-1})$.

Найдем полином a. Сначала проведем оценку степени полинома a. Пусть $\deg(a) = h$, тогда если θ_{n-1} — логарифмическая или экспоненциальная функция, то $\deg(a') = h$, а в остальных случаях $\deg(a') = h - 1$.

Рассмотрим три случая.

1) Если $P, \bar{c}_1, \bar{c}_2, a \in F(x)$, то $\deg(P) = h + \max(\deg(\bar{c}_1) - 1, \deg(\bar{c}_2))$. Отсюда следует, что $h = \deg(P) - \max(\deg(\bar{c}_1) - 1, \deg(\bar{c}_2))$. Если $h < 0$, то исходное подынтегральное выражение не интегрируется в элементарном виде. Если $h = 0$, то $a = \frac{P}{\bar{c}_2}$. Если $h > 0$, то $a = \sum_{i=0}^{h} a_i x^i$. Тогда равенство (8) примет вид:

$$P = \bar{c}_1 \left(a_0 + \sum_{i=1}^{h} i a_i x^{i-1} \right) + \bar{c}_2 \sum_{i=0}^{h} a_i x^i.$$

С помощью метода неопределенных коэффициентов можем найти $a_i \;\; i = 0, \ldots, h$.

2) Если θ_{n-1} — логарифмическая функция, то $\deg(P) = h + \max(\deg(\bar{c}_1), \deg(\bar{c}_2))$. Отсюда следует, что $h = \deg(P) - \max(\deg(\bar{c}_1), \deg(\bar{c}_2))$. Если $h < 0$, то исходное подынтегральное выражение не интегрируется в элементарном виде. Если $h \geqslant 0$, то $a = \sum_{i=0}^{h} a_i \theta_{n-1}^i$. Тогда равенство (8) примет вид:

$$P = \sum_{i=0}^{t} \bar{p}_i \theta_{n-1}^i = \bar{c}_1 \left(\sum_{i=0}^{h} a_i' \theta_{n-1}^i + \sum_{i=1}^{h} i a_i \theta_{n-1}' \theta_{n-1}^{i-1} \right) + \bar{c}_2 \sum_{i=0}^{h} a_i \theta_{n-1}^i.$$

Приравнивая левую и правую части данного равенства, получаем систему дифференциальных уравнений относительно $a_i \;\; i = 0, \ldots, h$, где $a_i \in F(x, \theta_1, \ldots, \theta_{n-2})$.

3) Если θ_{n-1} — экспоненциальная функция, тогда помимо старшей степени полинома a находим и младшую степень полинома a, так как $\exp^{-1}(x)$ тоже относится к полиномиальной части. Обозначим через $h1$ старшую степень полинома a, через $h2$ младшую степень полинома a, через $k1$ и $k2$ старшую и младшую степени полинома $\bar{c}_1$ соответственно, через $t1$ и $t2$ старшую и младшую степень полинома $\bar{c}_2$ и $m1$ и $m2$ старшую и младшую степень полинома P. Тогда $h1 = m1 - \max(k1, t1)$ и $h2 = m2 - \max(k2, t2)$. Отсюда получаем, что $a = \sum\limits_{i=h2}^{h1} a_i \theta_{n-1}^i$. Тогда равенство (8) примет вид:

$$P = \sum_{i=m2}^{m1} \bar{p}_i \theta_{n-1}^i = \bar{c}_1 \sum_{i=h2}^{h1} (a_i' + i a_i \mu') \theta_{n-1}^i + \bar{c}_2 \sum_{i=h2}^{h1} a_i \theta_{n-1}^i,$$

где μ — аргумент функции θ_{n-1}. Приравнивая левую и правую части полученного равенства, получаем систему дифференциальных уравнений относительно a_i, $i = h2, \ldots, h1$, где $a_i \in F_{n-2}$ и $F_{n-2} = F(x, \theta_1, \ldots, \theta_{n-2})$.

Пример 3. Рассмотрим интеграл:

$$\int e^{x^2} dx.$$

Обозначим функцию e^{x^2} через θ. Функция $\theta \in R(x, e^{x^2})$. По принципу Лиувилля, если данный интеграл существует, то его можно представить в виде:

$$\int \theta = B_1(x)\theta + \sum c_i \ln(v_i),$$

где $B_1 \in R(x), \; v_i \in R(x, \theta), \; c_i = const$. Продифференцируем обе части и получим:

$$\theta = (B_1' + 2xB_1)\theta + \sum c_i \frac{v_i'}{v_i}.$$

Отсюда следует равенство:

$$1 = B_1' + 2xB_1.$$

Так как левая часть равенства является полиномом от x, то B_1 тоже является полиномом от x. Пусть степень B_1 равна n, тогда $\deg(B_1') = n-1, \deg(2xB_1) = n+1$ и $\deg(1) = 0$. Получаем равенство $0 = n + 1$. Так как $n \geqslant 0$, то рассмотренный интеграл не берется в элементарном виде.

Пример 4. Рассмотрим интеграл

$$\int \frac{4x^2 + 4x - 1}{(x+1)^2} e^{x^2}.$$

Обозначим через $\theta = e^{x^2}$. По принципу Лиувилля, если данный интеграл существует, то его можно представить в виде:

$$\int f(\theta) = \int \frac{4x^2 + 4x - 1}{(x+1)^2} \theta = q_1 \theta, \quad q_1 \in Q(x).$$

Откуда получаем

$$\frac{4x^2+4x-1}{(x+1)^2} = q_1' + 2xq_1. \tag{9}$$

Так как левая часть равенства (9) — дробно-рациональная функция, то q_1 тоже будет дробно-рациональной функцией $q_1 = \frac{a}{b}$ и

$$\frac{4x^2+4x-1}{(x+1)^2} = \frac{a'b-ab'}{b^2} + 2x\frac{a}{b}.$$

Тогда $b = x+1$, $4x^2+4x-1 = a'b-ab'+2xab = (x+1)a' + (-1+2x^2+2x)a$. Пусть $\deg(a) = h$, тогда $h = \deg(4x^2+4x-1) - \deg 2x = 1$. Так как $h = 1$, то $a = a_1x + a_0$. Тогда получаем:

$$4x^2+4x-1 = a_1x + a_1 + (-1+2x^2+2x)(a_1x+a_0).$$

Приравниваем коэффициенты при степенях x, получаем $a_1 = 0$, $a_0 = 2$. Тогда

$$\int f = \frac{1}{x+1}e^{x^2}.$$

4 Заключение

Проблеме символьного интегрирования полиномиальной части интеграла была посвящена работа [9], но в ней не был рассмотрен случай, когда последнее расширение является экспоненциальной функцией. В данной статье представлен алгоритм интегрирования полиномиальной части интеграла $\int f$, когда $f \in F(x,\theta_1,\ldots,\theta_n)$, и последнее расширение θ_n является экспоненциальной функцией над $F_{n-1} = F(x,\theta_1,\ldots,\theta_{n-1})$. В данном алгоритме анализируются случаи, когда θ_{n-1} является экспонентой или логарифмической функцией, а также случай, когда $F_{n-1} = F(x)$. Рассмотрен метод неопределенных коэффициентов для нахождения полиномиальной части интеграла $\int f$, когда $f \in F(x,\theta_1,\ldots,\theta_n)$ и последнее расширение θ_n является логарифмической функцией над $F(x,\theta_1,\ldots,\theta_{n-1})$. Результатом этих исследований стала разработка в системе компьютерной алгебры Mathpar [17],[18] алгоритма символьного интегрирования смешанных функций.

ЛИТЕРАТУРА

1. *Geddes K., Czapor S., Labahn G.* Algorithms for computer algebra. Kluwer. 1992. Vol. 585.

2. *Панкратьев Е.В.* Элементы компьютерной алгебры (Конспекты спецкурса). М.:Механико-математический факультет МГУ.

3. *Risch R.H.* The problem of integration in finite terms // Trans. Amer. Math. Soc. 1969. P.167-189.

4. *Joel Moses.* Symbolic integration the stormy decade // Proceedings of the second ACM symposium on Symbolic and algebraic manipulation. 1971. P. 427-440.

5. *Дэвенпорт Д., Сирэ И., Турнье Э.* Компьютерная алгебра: Пер с франц. М.: Мир,1991.

6. *Trager B.* Integration of algebraic functions. PhD Thesis, MIT, 1984.

7. *Bronstein M.* The Transcendental Risch Differential Equation // J. Symbolic Comp. P. 49-60. 1990.

8. *Bronstein M.* Symbolic Integration Tutorial // ISSAC'98, Rostock.

9. *Тарарова С.М.* Символьное интегрирование композиций элементарных функций // Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 14. Вып. 1. 2009. С. 283-286.

10. *Rosenlicht M.* On Liouville's Theory of Elementary Function // Pacific J. Math. 1976. P. 485-492.

11. *Cherry G.W.* Integration in Finite Terms with Special Functions: the Error Function // J. Symbolic Comp. 1 P. 283-302. 1985.

12. *Kaltofen E.* A Note on the Risch Differential Equation // P. 359-366 in Proc. EUROSAM '84. Lecture Notes in Computer Science 174. ed. J. Fitch. Springer-Verlag 1984.

13. *Knowles P. H.* Integration of a Class of Transcendental Liouvillian Functions with Error-Functions. Part I // D'Youville College. U.S.A. 1988. P. 525-543.

14. *Rothstein M.* Aspects of Symbolic Integration and Simplification of Exponential and Primitive Functions. Ph.D. Thesis. Univ. of Wisconsin. Madison 1976. Vol. 119.

15. *Фихтенгольц Г.М.* Курс дифференциального и интегрального исчисления. Т. 1. М.: ФИЗМАТЛИТ, 2001.

16. *Фихтенгольц Г.М.* Курс дифференциального и интегрального исчисления. Т. 2. М.: ФИЗМАТЛИТ, 2001.

17. *Malaschonok G.I.* Project of Parallel Computer Algebra // Tambov University Reports. Series: Natural and Technical Sciences. V. 15. Issue 6. 2010. P. 1724-1729.

18. *Малашонок Г.И.* Компьютерная математика для вычислительной сети // Вестник Тамбовского университета. Сер.: Естественные и технические науки. Том 15. Вып. 1. 2010. С. 322-327.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке РФФИ (грант № 12-07-00755-а) и программы «Развитие потенциала высшей школы» (проект 2.1.1/10437).

Поступила в редакцию 20 февраля 2012 г.

TO THE PROBLEM OF CONSTRUCTING AN ALGORITHM FOR SYMBOLIC INTEGRATION

© **Svetlana Mikhailovna Tararova**

Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Mathematical Analysis Department, e-mail: tararovasveta@gmail.com

Key words: Liouville principle of the elementary integral, differential field, symbolic integration, polynomial part of the integral, elementary functions.

The work is devoted to the integration of the compositions of elementary functions. The integral is the sum of a polynomial and the rational part.Transcendental functions, which are included in these parts - is logarithmic andexponential functions of a differential field. We consider only the integral part of the polynomial. To solve this problem, the central role plays the Liouville theorem. These algorithms are used in the computer algebra system Mathpar.