УДК 512.7

Реализация алгоритма поиска базисов Гребнера идеалов колец многочленов от нескольких переменных на языке C++

Пивкин А. А., Сухарев Л. А.

Национальный исследовательский
Мордовский государственный университет им. Н. П. Огарёва

Аннотация: В статье рассматривается реализация алгоритма Бухбергера построения базиса Грёбнера идеала кольца многочленов от нескольких переменных над полем комплексных чисел на языке программирования C++. Известная теорема Гильберта о базисе идеала кольца многочленов от нескольких переменных обеспечивает существование конечного базиса такого идеала. В случае задания идеала конечным набором многочленов с помощью алгоритма Бухбергера можно построить базис Грёбнера этого идеала. В работе подробно изложена компьютерная реализация одной из версий алгоритма при условии лексикографического упорядочивания многочленов. В частности рассмотрены: представление многочленов от нескольких переменных с комплексными коэффициентами в компьютере, задание арифметических операций над ними, исследование зацеплений двух многочленов базиса, реализация операции редуцирования и минимизация базиса Грёбнера. Приведены примеры работы программы и комментарии основных моментов программного кода.

Ключевые слова: базис Грёбнера, алгоритм Бухбергера.

В настоящей работе мы будем придерживаться обозначений и терминологии, используемых в [1]. Кольцо многочленов от n переменных над полем $\mathbb{C}$ стандартно обозначим через $\mathbb{C}[x_1,\ \ldots, x_n]$, а его идеал I, порожденный многочленами $f_1,\ \ldots f_k$, обозначим $\langle f_1,\ \ldots f_k\rangle$, при этом запись $I = \langle f_1,\ \ldots f_k\rangle$ означает, что идеал I состоит из всевозможных линейных комбинаций многочленов $f_1,\ \ldots f_k$ с коэффициентами из кольца многочленов $\mathbb{C}[x_1,\ \ldots, x_n]$. Для программной реализации на языке программирования C++ построения базиса Грёбнера идеала I требуются задание арифметических операций и вычисление зацеплений двух многочленов.

Каждому одночлену поставим в соответствие пару, состоящую из набора его степеней и коэффициента. Поскольку многочлен – это сумма одночленов, то в качестве структуры для многочлена будем использовать ассоциативный контейнер `map`, который работает по принципу «ключ – значение» [2]. Это означает, что, во-первых, при добавлении новых элементов в контейнер он будет отсортирован по возрастанию, т. е. одночлены будут упорядочены лексикографически; а во-вторых, если возникает необходимость добавить существующий элемент, то достаточно изменить его значение `map`.

Реализуем операции сложения и умножения над многочленами с комплексными коэффициентами следующим образом:

```
map<vector<int>, complex<double» operator+(const map<vector<int>,
complex<double»& f, const map<vector<int>, complex<double»& g) {

    map<vector<int>, complex<double» h = g;
```

```
    vector<vector<int»del;
    for (auto &x : f) {
        h[x.first] += x.second;
        if (fabs(h[x.first].real()) + fabs(h[x.first].imag()) == 0.0)
        del.push_back(x.first);
    }
    for (auto& x : del) {
        h.erase(x);
    }
    return h;
}

map<vector<int>, complex<double» operator*(const map<vector<int>,
complex<double»& f, const map<vector<int>, complex<double»& g) {
    map<vector<int>, complex<double» h;
    for (auto &x : f)
        for (auto &y : g)
           h[x.first + y.first] += x.second * y.second;
    return h;
}
```

Поскольку при сложении двух многочленов после приведения подобных некоторые одночлены будут иметь коэффициент, равный нулю, то эти одночлены не полежат хранению, поэтому в реализации сложения создается массив `del`, в который записываются удаляемые наборы степеней с нулевыми коэффициентами.

Вычисление зацеплений в алгоритме Бухбергера сводится к нахождению S-многочлена двух многочленов, который определяется формулой

$$S(f,\ g) = \frac{LCM(LM(f),\ \ LM(g))}{LT(f)} \cdot f - \frac{LCM(LM(f),\ \ LM(g))}{LT(g)} \cdot g.$$

Коэффициенты при f и g всегда являются одночленами, поэтому никакого деления проводить не требуется [1], [3]. Определим функцию `S`, которая в качестве параметров принимает два многочлена f и g, и возвращает их S-многочлен.

```
map<vector<int>, complex<double» S(map<vector<int>,
complex<double»& f, map<vector<int>, complex<double»& g) {
    map<vector<int>, complex<double» q1, q2;
    vector<int>lcm = LCM(f.rbegin()->first, g.rbegin()->first);
    q1[lcm - f.rbegin()->first] = 1.0 / f.rbegin()->second;
    q2[lcm - g.rbegin()->first] = -1.0 / g.rbegin()->second;
    q1 = f * q1;
    q2 = g * q2;
    return (q1 + q2);
```

```
}
```

Реализация операции редукции представлена функцией `reduction`:

```
void reduction( vector<map<vector<int>, complex<double»>& basis,
map<vector<int>, complex<double»& f)
{

    auto it = f.rbegin();
    bool flag = false;
    for (; it != f.rend();)
    {

        for (int i = 0; i < basis.size(); i++) {
           if (division(basis[i].rbegin()->first ,it->first)) {
              map<vector<int>, complex<double» q;
              q[it->first - basis[i].rbegin()->first] +=
              -it->second /basis[i].rbegin()->second;
              f = f + (q * basis[i]);
              flag = true;
              break;
           }
        }
        if (f.size() == 0) {
           break;
        }
        if (flag)
        {
        it = f.rbegin();
        flag = false;
        }
        else
        it++;

    }

}
```

В качестве параметров функции `reduction` передается текущий набор многочленов `basis` и многочлен `f`, который необходимо редуцировать. Первый цикл проходит по всем одночленам многочлена f, а второй проходит по набору многочленов. Если одночлен редуцируемого многочлена делится на старший одночлен некоторого многочлена, то производим операцию редукции, а затем проходим по всем одночленам многочлена f с самого начала.

Реализация алгоритма Бухбергера при этом проведена следующим образом:

```
void Buchberger_algorithm(vector<map<vector<int>, complex<double»>&
basis) {

    int k = 0;
    while (k < basis.size())
    {
```

```
        int j = 0;
        while (j < basis.size())
        {
            if (j == k) {
            j++;
            continue;
            }
            if (!check(basis[k], basis[j])) {
                j++;
                continue;
            }
            map<vector<int>, complex<double» S_fg;
            S_fg = S(basis[k], basis[j]);
            j++;
            if (!S_fg.size())
                continue;
            reduction(basis, S_fg);
            if (S_fg.size())
                basis.push_back(S_fg);
        }
        k++;
    }

}
```

Функция `Buchberger_algorithm`, использующаяся в построении базиса Грёбнера, в качестве входных параметров принимает набор многочленов `basis`. Двойным циклом по всему набору многочленов она проверяет с помощью функции `check(f, g)` имеют ли два многочлена зацепление. Если имеют, то функция находит их S-многочлен и редуцирует его. Если он не редуцируется к нулю, то функция добавляет его в набор базисных многочленов.

Также в программе реализована возможность пошагового вывода работы алгоритма.

Ниже приведены примеры работы программы.

Пример 1.

Идеал: $I = \langle x_1^2 - 1,\ \ x_1x_2 - x_2,\ \ x_1x_3 + x_3\rangle$.

Файл `input.txt`:

```
Число_переменных: 3
Число_многочленов: 3
Вывод: true
(1;0)x1^2+(-1;0);
(1;0)x1 x2 +(-1;0)x2;
(1;0)x1 x3 +(1;0)x3;
```

Файл `output.txt`:

```
Многочлены (1; 0) x1^2 + (-1; 0) и (1; 0) x1 x2 + (-1; 0) x2 имеют зацепление
F_1_2=(1; 0) x1 x2 + (-1; 0) x2
-- > редукция с помощью многочлена f_2=(1; 0) x1 x2 + (-1; 0) x2 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (1; 0) x1^2 + (-1; 0) и (1; 0) x1 x3 + (1; 0) x3 имеют зацепление
F_ 1_3=(-1; 0) x1 x3 + (-1; 0) x3
-- > редукция с помощью многочлена f_3=(1; 0) x1 x3 + (1; 0) x3 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (1; 0) x1 x2 + (-1; 0) x2 и (1; 0) x1^2 + (-1; 0) имеют зацепление
F_2_1=(-1; 0) x1 x2 + (1; 0) x2
-- > редукция с помощью многочлена f_2=(1; 0) x1 x2 + (-1; 0) x2 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (1; 0) x1 x2 + (-1; 0) x2 и (1; 0) x1 x3 + (1; 0) x3 имеют зацепление
F_2_3=(-2; 0) x2 x3
Больше зацеплений нет.
Добавим f_4=(-2; 0) x2 x3 в набор.

Многочлены (1; 0) x1 x2 + (-1; 0) x2 и (-2; 0) x2 x3 имеют зацепление
F_2_4=(-1; 0) x2 x3
-- > редукция с помощью многочлена f_4=(-2; 0) x2 x3 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (1; 0) x1 x3 + (1; 0) x3 и (1; 0) x1^2 + (-1; 0) имеют зацепление
F_3_1=(1; 0) x1 x3 + (1; 0) x3
-- > редукция с помощью многочлена f_3=(1; 0) x1 x3 + (1; 0) x3 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (1; 0) x1 x3 + (1; 0) x3 и (1; 0) x1 x2 + (-1; 0) x2 имеют зацепление
F_3_2=(2; 0) x2 x3
-- > редукция с помощью многочлена f_4=(-2; 0) x2 x3 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (1; 0) x1 x3 + (1; 0) x3 и (-2; 0) x2 x3 имеют зацепление
F_3_4=(1; 0) x2 x3
-- > редукция с помощью многочлена f_4=(-2; 0) x2 x3 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (-2; 0) x2 x3 и (1; 0) x1 x2 + (-1; 0) x2 имеют зацепление
F_4_2=(1; 0) x2 x3
-- > редукция с помощью многочлена f_4=(-2; 0) x2 x3 -- > 0
S-многочлен редуцируется к нулю.

Многочлены (-2; 0) x2 x3 и (1; 0) x1 x3 + (1; 0) x3 имеют зацепление
F_4_3=(-1; 0) x2 x3
-- > редукция с помощью многочлена f_4=(-2; 0) x2 x3 -- > 0
```

```
S-многочлен редуцируется к нулю.

Базис Гребнера:
(1; 0) x1^2 + (-1; 0)
(1; 0) x1 x2 + (-1; 0) x2
(1; 0) x1 x3 + (1; 0) x3
(-2; 0) x2 x3

Минимальный редуцированный базис Грёбнера
(1; 0) x1^2 + (-1; 0)
(1; 0) x1 x2 + (-1; 0) x2
(1; 0) x1 x3 + (1; 0) x3
(1; 0) x2 x3
```

Таким образом минимальный редуцированный базис Грёбнера имеет вид:

$$\{x_1^2 - 1, \quad x_1x_2 - x_2, \quad x_1x_3 + x_3, \; x_2x_3\}.$$

Пример 2.
Идеал: $I = \langle x_1^2 + 2i\ x_4, \quad x_1x_2 - x_2, \quad x_1x_3 + x_3, \quad x_1x_3 + 4i\ x_3^4\rangle$.

Файл `input.txt`:

```
Число_переменных: 4
Число_многочленов: 4
Вывод: false
(1;0) x1^2+(0;2) x4;
(1;0) x1 x2 +(-1;0) x2;
(1;0) x1 x3 +(1;0) x3;
(1;0) x1 x3 +(0;4) x3^4;
```

Файл `output.txt`:

```
Минимальный редуцированный базис Грёбнера
(1; 0) x1^2 + (0; 2) x4
(1; 0) x1 x2 + (-1; 0) x2
(1; 0) x1 x3 + (1; 0) x3
(1; 0) x2 x4 + (0; -0.5) x2
(1; 0) x3 x4 + (0; -0.5) x3
(1; 0) x3^4 + (0; 0.25) x3
(1; 0) x2 x3
```

Таким образом минимальный редуцированный базис Грёбнера имеет вид:

$$\{x_1^2 - 2ix_4, \quad x_1x_2 - x_2, \quad x_1x_3 + x_3, \quad x_2x_4 - 0.5ix_2, \; x_3x_4 - 0.5x_3, \; x_3^4 + 0.25x_3, \quad x_2x_3\}.$$

Стоит отметить, что известны более эффективные алгоритмы поиска базисов Гребнера идеалов, например, F4 [4] и F5 [5]. В качестве явного преимущества представленная в настоящей работе программная реализация алгоритма Бухбергера поиска базиса Грёбнера идеала кольца многочленов от нескольких переменных с комплексными коэффициентами имеет простоту изложения, доступного для понимания студентами.

Суть работы состоит в изложении методологического подхода, позволяющего связать задачу фундаментальной алгебры с ее решением с помощью компьютерных средств и современного программного обеспечения.

Литература

1. Аржанцев И.В. Базисы Грёбнера и системы алгебраических уравнений. Москва: МЦНМО, 2003. 68 с.
2. Галовиц Я. C++17 STL. Стандартная библиотека шаблонов. Санкт-Петербург: Питер, 2018. 432 с.
3. Кокс Д., Литтл Дж., О'Ши Д. Идеалы, многообразия и алгоритмы. Введение в вычислительные аспекты алгебраической геометрии и коммутативной алгебры. Пер. с англ. Москва: Мир, 2000. 687 с.
4. Faugere J.-C. A new efficient algorithm for computing Grobner bases (F4) // Journal of Pure and Applied Algebra. North-Holland: Elsevier. 1999. V. 139. P. 61–88.
5. Faugere J.-C. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5) // Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation. New York: ACM. 2002. P. 75–83.

MSC 13A15

Implementation of the algorithm for finding Groebner bases of polynomial rings ideals from several variables in C++

A. A. Pivkin , L. A. Sukharev

National Research Ogarev Mordovia State University

Abstract: The article is devoted to the implementation of the Buchberger algorithm for constructing the Groebner basis of the polynomial ring ideal of several variables over a field of complex numbers in the C++ programming language. Hilbert's well-known theorem on the basis of the polynomials ring ideal in several variables ensures the existence of a finite basis for such an ideal. The well-known Buchberger algorithm constructs the Groebner basis of this ideal In the case of specifying an ideal by a finite set of polynomials. The paper describes in detail the computer implementation of one version of the algorithm under the condition of lexicographic ordering of polynomials. In particular, the following are considered: the representation of polynomials from several variables with complex coefficients in a computer, the assignment of arithmetic operations on them, the study of the meshing of two basis polynomials, the implementation of the reduction operation and the minimization of the Grobner basis. Examples of the program's operation and comments on the main points of the program code are given.

Keywords: Gröbner basis, Buchberger's algorithm.

References

1. I.V. Arzhantsev, Gröbner Bases and Systems of Algebraic Equations, 3rd edn, Moscow, MCCMO, 2003. (in Russian)

2. Ya. Galovits, C++17 STL. Standard Template Library, St. Petersburg, Peter, 2018, 432 p.

3. D. Cox, J. Little, D. O'Shea, Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, Undergraduate Texts in Mathematics, Moscow, Mir, 2000.

4. J.-C. Faugere, A new efficient algorithm for computing Grobner bases (F4), *Journal of Pure and Applied Algebra. North-Holland: Elsevier*, 1999, V. 139, P. 61–88.

5. J.-C. Faugere, A new efficient algorithm for computing Gröbner bases without reduction to zero (F5), *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation. New York: ACM*, 2002, P. 75–83.