# ginv Руководство пользователя

*Release 1.9*

Блинков Ю. А. и Гердт В. П.

24 апреля 2008 г.

BlinkovUA@info.sgu.ru и gerdt@jinr.ru

**Аннотация**

Представлена система компьютерной алгебры GINV (сокращение от Gröbner INVolutive), разработанная для исследования и решения систем алгебраических, дифференциальных и разностных уравнений полиномиального типа методами их приведения в инволюцию. В основе системы лежат авторские алгоритмы построения инволютивных базисов Жане и базисов типа Жане для полиномиальных идеалов и модулей, а также приведенных базисов Грёбнера. GINV состоит из библиотеки программ на языке C++, являющейся модулем языка Python, доступна на сайте и распространяется на условиях GPL v2.0 сайта http://invo.jinr.ru/ginv/.

# Содержание

# 1 Введение

Проект **GINV** представляет собой специализированную систему компьютерной алгебры для систем полиномиальных уравнений, модулей, дифференциальных уравнений и линейных разностных уравнений (разностных схем). Она состоит из библиотеки программ на языке **C++** и модуля языка **Python**. Программа распространяется на условиях GPL v2.0 с сайта http://invo.jinr.ru/ginv/. **GINV** используется в качестве внешнего библиотечного модуля пакетом *Janet* [3] написанном на языке универсальной системы компьютерной алгебры **Maple** и доступного на сайте http://wwwb.math.rwth-aachen.de/Janet/.

Из внешних библиотек используется кроссплатформенная библиотека **gmp** для арифметики целых, рациональных и чисел с плавающей точкой произвольной точности. Документация системы **GINV** состоит из “руководства пользователя” и “руководства разработчика”. Первая из них построена с помощью стандартных средств для документирования модулей языка **Python**, а вторая с помощью системы **Doxygen**. Документация поддерживается на двух языках, русском и английском, и доступна на сайте http://invo.jinr.ru/ginv/ в форматах *pdf* и *html*. Руководство разработчика содержит более 1000 страниц формата A4.

Алгоритмы построения базисов Грёбнера имеют экспоненциальную сложность построения как по времени выполнения, так и по требуемой памяти. Поэтому, при реализации этих алгоритмов бывает трудно соблюсти их универсальность, не забывая при этом об эффективности. Как показывает сравнение **GINV** [15] с реализацией на **C** [12] этого удалось достичь. Сравнение результатов расчета с последней версией **GINV** доступны на вышеуказанном сайте.

Следует отметить, что в последние годы развитие системы **GINV** осуществляется в сотрудничестве с Техническим университетом г. Аахен, Германия. Сотрудниками этого университета было, в частности, встроено в систему работа с алгебраическими расширениями поля $\mathbb{Q}$ и классический алгоритм Брауна [5] для вычисления наибольшего общего делителя, требуемый для вычислений в кольце многочленов над полем $\mathbb{Q}(p_1, \ldots, p_k)$ в случае параметрических коэффициентов.

## 1.1 Текущая версия 1.9

Значительно переделано данное руководство.

Вышла статья [4] посвященная описанию **GINV**.

Daniel Robertz[1] добавил следующие реализации (см. 5.1) для работы с расширениями алгебраического поля *AlgebraicFieldExtGmpQ*, *AlgebraicFieldExtModularShort* и *AlgebraicFieldExtNParameter*.

Sebastian Jambor[2] сделал эффективную реализацию алгоритма вычисления наибольшего общего делителя для нескольких переменных. Это позволило добавить в **GINV** следующие классы коэффициентов (см. 5.1) *OneParametrGmpZ*, *OneParametrGmpZ*, *NParameterGmpZ* и *NParameterField*

Юрий Блинков[3] при помощи Daniel Robertz добавил возможность организации вычислений над кольцом (3). В настоящие время имеется реализация только для (см. 5.1) *GmpZRing*.

## 1.2 Версия 1.2

Первая версия выложенная на сайте http://invo.jinr.ru/ginv/.

[1] daniel@momo.math.rwth-aachen.de
[2] sebastian@momo.math.rwth-aachen.de
[3] BlinkovUA@info.sgu.ru

# 2 Быстрый старт

## 2.1 Установка

Для установки необходимо иметь установленный в операционной системе *Python* с версией (>=2.4). Скачать *Python* можно с сайта http://www.python.org.

Для запуска `ginv` после установки в нестандартный каталог (например не имея прав администратора) необходимо определить переменную окружения PYTHONPATH. PYTHONPATH представляет собой список каталогов (формат такой же, как у строки PATH), разделенных символом ':' ("Unix") или ';' ("Windows"), в которых будет производится поиск модулей перед поиском по умолчанию.

### Windows

Для установки "Windows" используется уже скомпилированный модуль `ginv`. Это накладывает некоторые ограничения на тип и разрядность используемого процессора. Модуль `ginv` собран с кодом *i586*. В результате возможно значительное снижение производительности, например при использовании 64-битого процессора.

Сборка для "Windows" аналогична сборки под "Unix".

### Unix

Для инсталляции под "Unix" требуется библиотека *gmp* http://www.swox.com/gmp собранная с поддержкой *C++*. Для оптимизация кода под архитектуру компьютера можно внести корректировку опций компилятора *C++* в файле 'setup.py'.

Сборка модуля `ginv` происходит по стандартной схеме для Python

```
python setup.py build
```

Инсталляция в стандартный каталог (и сборка если модуль не был собран)

```
python setup.py install
```

Для установки в нестандартный каталог

```
python setup.py install --prefix=/home/user/pyginv
```

## 2.2 Запуск

Для проверки правильности установки достаточно запустив Python импортировать модуль командой

```
>>> import ginv
```

Следующий пример позволяет привести систему полиномиальных уравнений к "треугольному виду" и вывести ее на экран

```
import ginv

st = ginv.SystemType("Polynomial")
im = ginv.MonomInterface("Lex", st, ['x', 'y', 'z'])
```

```
ic = ginv.CoeffInterface("GmpZ", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
iD = ginv.DivisionInterface("Janet", iw)

basis = ginv.basisBuild("TQ", iD,   ['x^3 - y^2 + z - 1',
  'y^3 - z^2 + x - 1',   'z^3 - x^2 + y - 1'])

for p in basis.iterIB():
  print p


print "hilbertPolynomial = ", basis.hilbertPolynomial()
```

Результатом работы программы будет вывод на экран трех элементов приведенного базиса Грёбнера и многочлена Гильберта:

```
139536745252397484708849 6*x + (-369740120437206066027 47)*z^26 +
(-180423871499814059499 31)*z^25 + 330306424279887894832 93*z^24 +
290848876458135913225112*z^23 + 129076392042710083597936*z^22 +
(-219952910543490007651408)*z^21 + (-735043497452966299206159)*z^20 +
(-310272794617140912662271)*z^19 + 160633475001590756076807*z^18 +
982342916167428059233558*z^17 + (-209204326958442264627 72)*z^16 +
123787951244996878821776 1*z^15 + (-649364739492104109294111)*z^14 +
177078379825139598960064 0*z^13 + (-326957078565586511526 7646)*z^12 +
101713739257354856916042 5*z^11 + (-538029066871643242248 0115)*z^10 +
251131841492531327135873 6*z^9 + (-582076592674322219851 4292)*z^8 +
 391623410500680942536214 4*z^7 + (-127174098696048164937 4276)*z^6 +
452657186612834476344576 8*z^5 + 422539940712448006464 7579*z^4 +
(-170248813991134662305 4830)*z^3 + 259932158820846278782 8165*z^2 +
(-649905019834301035687 7518)*z + 113853164103545382507 1656
139536745252397484708849 6*y + (-242632459056334226660 89)*z^26 +
(-256410179757740270610 5)*z^25 + (-761271693439804295251 3)*z^24 +
183708473843508669676888*z^23 + 273825904930647065617 28*z^22 +
933828836590525029178 72*z^21 + (-445019735963407736728965)*z^20 +
(-124580325693827052967653)*z^19 + (-496913514381305731286115)*z^18 +
686028568407960758274770*z^17 + (-293264534395468068345180)*z^16 +
175491631773986965591308 3*z^15 + (-149416104442071932675814 9)*z^14 +
316686484866592249150716 8*z^13 + (-432414886671810736671 9898)*z^12 +
511819335382059435434046 7*z^11 + (-966998804194819875992 5505)*z^10 +
823496439253093918796118 4*z^9 + (-148002909861464240156702 04)*z^8 +
144077097103726415422746 56*z^7 + (-171102385581693093055355 00)*z^6 +
192898551955674456532912 24*z^5 + (-140716752486852970911562 15)*z^4 +
172084953165649105415404 22*z^3 + (-978026103231559587116823 3)*z^2 +
568454360708997967244325 4*z + (-460643075780459538920 4888)
z^27 + (-9)*z^24 + 29*z^21 + 6*z^19 + (-53)*z^18 + 22*z^17 + (-63)*z^16 +
96*z^15 + (-149)*z^14 + 242*z^13 + (-261)*z^12 + 484*z^11 + (-545)*z^10 +
740*z^9 + (-908)*z^8 + 972*z^7 + (-1220)*z^6 + 1047*z^5 + (-1045)*z^4 +
943*z^3 + (-535)*z^2 + 422*z + (-216)
hilbertPolynomial = 27
```

Структура многочлена Гильберта показывает, что число корней системы с учетом их кратности равно 27.

# 3 Виды входных данных

Виды входных данных определяют способы задания полиномов и правила работы с ними. Различаются 4 вида: полиномы, модули, дифференциальные уравнения и разностные уравнения. Основное различие их в форме представления мономов. Полиномы могут также иметь различные виды коэффициентов: числовые и параметрические. Входные данные могут представляться в любых допустимых выражениях, в том числе и с дробями. Дробь должна иметь в знаменателе тип данных принадлежащих коэффициенту, который после приведения к общему знаменателю отбрасывается. Тип системы задается классом `SystemType`. При разборе учитывается обычный приоритет операций. Допустимые в **GINV** выражения можно представить в форме БНФ (Бэкуса – Наура Форма):

$$
\begin{aligned}
\langle ICoeff \rangle &::= \langle \text{положительное} \rangle \mid \langle \text{параметр} \rangle \\
\langle IMonom \rangle &::= \langle \text{зависимая} \rangle \mid \langle \text{независимая} \rangle \\
\langle IPoly \rangle &::= \langle ICoeff \rangle \mid \langle IMonom \rangle \\
\langle IExpression \rangle &::= \langle IPoly \rangle \\
\langle IExpression \rangle &::= \langle IPoly \rangle \text{ '/' } \langle ICoeff \rangle \\
\langle IExpression \rangle &::= \text{'-' } \langle IExpression \rangle \\
\langle IExpression \rangle &::= \langle IExpression \rangle \text{ '+' } \langle IExpression \rangle \\
\langle IExpression \rangle &::= \langle IExpression \rangle \text{ '-' } \langle IExpression \rangle \\
\langle IExpression \rangle &::= \langle IExpression \rangle \text{ '*' } \langle IExpression \rangle \\
\langle IExpression \rangle &::= \langle IExpression \rangle \text{ '/' } \langle ICoeff \rangle \\
\langle IExpression \rangle &::= \text{'-' } \langle IExpression \rangle \\
\langle IExpression \rangle &::= \langle IExpression \rangle \text{ '^' } \langle \text{положительное} \rangle \\
\langle IExpression \rangle &::= \text{'df('} \langle IExpression \rangle \text{','} \langle \text{зависимая} \rangle \text{')'} \\
&\quad \mid \text{'df('} \langle IExpression \rangle \text{','} \langle \text{зависимая} \rangle \text{','} \langle \text{положительное} \rangle \text{')'} \\
\langle IExpression \rangle &::= \text{'T('} \langle IExpression \rangle \text{','} \langle \text{зависимая} \rangle \text{')'} \\
&\quad \mid \text{'T('} \langle IExpression \rangle \text{','} \langle \text{зависимая} \rangle \text{','} \langle \text{положительное} \rangle \text{')'} \\
\langle IExpression \rangle &::= \text{'['} \langle IExpression \rangle [\text{','} \ < IExpression > ]\text{']'}
\end{aligned}
$$

Выше использованы обозначения:
⟨положительное⟩ – целое положительное число,
⟨независимая⟩ – независимая переменная,
⟨зависимая⟩ – зависимая переменная,
'df' – производная,
'T' – оператор сдвига,
'[' . . . ']' – модуль.

**class** **SystemType**(*type, module=None, rightPart=None*)
Определяет тип полинома

**type:** строка определяющая тип полинома

| Значение type | Тип системы |
|---|---|
| `"Polynomial"` | Полиномиальная система 3.1 |
| `"DifferentialEquation"` | Система линейных дифференциальных уравнений 3.3 |
| `"FiniteDifferenceScheme"` | Система линейных разностных уравнений 3.4 |

**module:** целое число задающая размерность модуля

**rightPart:** целое число задающая размерность правых частей равенства, на которые будут получены соотношения определяющие совместность системы, имеет смысл только для модуля.

Класс `SystemType` имеет следующие атрибуты:

**type**
: строка определяющая тип системы (read-only)

**module**
: целое число задающая размерность модуля (read-only)

**rightPart**
: целое число задающая размерность правых частей равенства (read-only)

## 3.1 Полиномы

Например полином от переменных `['x','y','z']` может представлять собой строку следующего вида

```
p = 'df((x^3+y)^2, x, y, 2)  + ((1+x)+y*z*y)*(z^3-9)'
```

В данном выражениии `df` представляет собой производную по переменным $x$ и дважды по $y$.

## 3.2 Модули

Модули представляют собой представление ввиде списка полиномов

```
m = ['(x^3+y)^2', '1+x', 'y*z*y*(z^3-9)', '0', '1']
```

## 3.3 Дифференциальные уравнения

Дифференциальное уравнение от зависимых переменных `['u','v']` и независимых переменных `['x','y','z']` может представлять собой строку следующего вида

```
d = 'df(x*u, x, 2) + 3*df(v, y, y, y)*y - u + y'
```

## 3.4 Разностные уравнения

Разностное уравнение от зависимых переменных `['u','v']`, независимых переменных `['t','x']` и параметров `['tau','h']` может представлять собой строку следующего вида

```
f = 'T(u, t, 2, x)*h + (T(v,x) - T(v, x, 3))*tau'
```

# 4 Мономы

В `ginv` реализованы следующие виды упорядочений:

**Упорядочения, совместимые с полной степенью:** `"TopDegRevLex"`, `"DegRevLex"`

**Упорядочения с векторизацией вычислений:** `"TopDegRevLexByte"`, `"DegRevLexByte"`

**Note:** Работа с машинным словом наиболее быстрая для процессора. Векторизация позволяет разместить в машинном слове (для компиляторов *C* это тип *unsigned int*) несколько показателей степеней. Тем самым в несколько раз ускоряются некоторые операции с мономом и уменьшаются затраты на память. Это имеет смысл когда показатели степеней ограничены и целесообразно только для упорядочений, совместимых с полной степенью.

Для векторизации необходимо знать где в данном компьютере расположен самый значимый бит: в начале или в конце машинного слова. Это задается макросами *C WORDS_BIGENDIAN* или *WORDS_LITTLEENDIAN*. Если они не заданны, то векторизация решает только задачу уменьшения используемой памяти.

**Упорядочения, несовместимые с полной степенью:** `"TopLex"`, `"TopElim"`, `"PotLex"`, `"PotDegRevLex"`, `"PosElim"`, `"Lex"`, `"Elim"`

## 4.1 Интерфейс

**class `MonomInterface`**(*order, systemType, independ, depend=None, varSep=None*)
Задает правила работы с мономом

**`order`:** строка определяющая тип упорядочения

| Значение `order` | Тип упорядочения |
|---|---|
| `"TopDegRevLex"` | упорядочение для модулей сначала по полной степени, а затем по обратной лексикографии и с "термом старше позиции" |
| `"DegRevLex"` | упорядочение сначала по полной степени, а затем по обратной лексикографии |
| `"TopDegRevLexByte"` | упорядочение для модулей сначала по полной степени, а затем по обратной лексикографии и с "термом старше позиции"и с использованием векторизации |
| `"DegRevLexByte"` | упорядочение сначала по полной степени, а затем по обратной лексикографии с использованием векторизации |
| `"TopLex"` | упорядочение для модулей по лексикографии и с "термом старше позиции" |
| `"TopElim"` | исключающее упорядочение с "термом старше позиции" |
| `"PotLex"` | упорядочение для модулей по лексикографии и с "позицией старше терма" |
| `"PotDegRevLex"` | упорядочение для модулей сначала по полной степени, а затем по обратной лексикографии и с "позицией старше терма" |
| `"PosElim"` | исключающие упорядочение для модулей с "позицией старше терма" |
| `"Lex"` | лексикографическое упорядочение |
| `"Elim"` | исключающее упорядочение |

**systemType:** определяет тип системы (см. 3)

**independ:** список независимых переменных

**depend:** список зависимых переменных, имеет смысл только для упорядочений поддерживающих модули

**varSep:** номер разделяющей переменной (зависимой или независимой), имеет смысл только для исключающих упорядочений

Класс `MonomInterface` имеет следующие методы:

**order**()
возвращает строку определяющую тип упорядочения

**dimIndepend**()
возвращает количество независимых переменных

**independ**()
возвращает список независимых переменных

**dimDepend**()
возвращает количество зависимых переменных

**depend**()
возвращает список зависимых переменных

**varSep**()
возвращает номер разделяющей переменной (зависимой или независимой)

## 4.2 Моном

**class Monom**(*monomInterface, monom*)
Задает мономом

**monomInterface:** определяет интерфейс монома (см. 4.1)

**monom:** строка инициализирующая моном

| Пример значения monom | Тип системы |
|---|---|
| `'1'` | единичный моном для любого типа системе |
| `'x*yˆ4*x'` | полиномиальная система 3.1 |
| `'df(u, x, y, 4, x)'` | система линейных дифференциальных уравнений 3.3 |
| `'T(u, x, y, 4, x)'` | система линейных разностных уравнений 3.4 |

Класс `Monom` имеет следующие методы:

**degree**()
возвращает сумму степеней переменных монома

**dependVar**()
возвращает номер зависимой переменной

**setZero**()
делает степени переменных монома равными нулю

**prolong**(*var, deg=1*)
умножает моном на переменную с номером *var* в степени *deg*

**gcd**(*monom*)
возвращает сумму степеней переменных НОД монома и монома *monom*.

Мономы должны иметь одинаковый интерфейс.

**lcm**(*monom*)
возвращает НОК монома и монома *monom*.

Мономы должны иметь одинаковый интерфейс.

**divisibility**(*monom*)
возвращает `True` если моном делится на моном *monom*, иначе `False`.

Мономы должны иметь одинаковый интерфейс.

**divisibilityTrue**(*monom*)
возвращает `True` если моном делится на моном *monom* и они не равны друг другу, иначе `False`.

Мономы должны иметь одинаковый интерфейс.

Класс `Monom` может быть аргументом следующих функций:

**str**(*monom*)
возвращает представление в виде строки монома *monom* согласно типу системы 3.

Аналогично работает команда Python **print**

**cmp**(*monom1, monom2*)
возвращает 1 если `monom1 > monom2`,
возвращает 0 если `monom1 == monom2`,
возвращает `-1` если `monom1 < monom2`.

Мономы сравниваются согласно определенному типу упорядочения. Мономы должны иметь одинаковый интерфейс.

**<, >, <=, >=, ==, !=**(*monom1, monom2*)
Для этих операций мономы сравниваются согласно определенному типу упорядочения. Мономы должны иметь одинаковый интерфейс.

Моном может быть использован в логических выражениях. Единичный моном дает в логических выражениях `False`, а остальные `True`

***, /**(*monom1, monom2*)
Эти операции возвращают произведение и частное мономов. Для выполнения деление *monom1* должен делится на *monom2*.

Мономы должны иметь одинаковый интерфейс.

***=**(*monom1, monom2*)
Эти операции присваивает произведение мономов переменной *monom1*.

Мономы должны иметь одинаковый интерфейс.

**len**(*monom*)
возвращает число независимых переменных в интерфейсе монома *monom*.

Аналогична методу `dimIndepend` класса `MonomInterface`.

**[]**(*monom, i*)
возвращает степень *i* независимой переменной в мономе *monom*.

Моном представляет собой итератор языка Python:

```
import ginv

st = ginv.SystemType("Polynomial")
im = ginv.MonomInterface("DegRevLex", st, ['x', 'y', 'z'])
m = ginv.Monom(im, "x^3*y*x*z^2")

for d in m: print d,
```

В результате будет распечатан список степеней '`4 1 2`'.

# 5 Коэффициенты

В `ginv` реализованы следующие виды коэффициентов:

**Коэффициенты многочлена образуют кольцо:**

```
"GmpZ"
"GmpZRing"
"OneParametrModularShort"
"TwoParametrModularShort"
"OneParametrGmpZ"
"TwoParametrGmpZ"
```

**Коэффициенты многочлена образуют поле:**

```
"IGmpQ"
"IModularShort"
"AlgebraicFieldExtGmpQ"
"AlgebraicFieldExtModularShort"
"AlgebraicFieldExtNParameter"
"NParameterField"
```

**Note:** В случае поля некоторые операции над многочленом можно сделать более эффективными, например в модулярном случае, если это не требует вычисления НОД.

Для организации приведения многочленов, из-за присутствия операции деления в ее определении

$$p \underset{f}{\longrightarrow} g = p - (\mathrm{lc}(p)/\mathrm{lc}(f))\, f \times v, \tag{1}$$

нужно дополнительно определить способ работы с коэффициентами, образующими кольцо. Здесь использованы обозначения $\mathrm{lm}(f)$ и $\mathrm{lc}(f)$ — старший моном и его коэффициент относительно используемого допустимого порядка на мономах. Существуют два способа:

$$p \underset{f}{\longrightarrow} g = \mathrm{lc}(f)/\gcd(\mathrm{lc}(p),\mathrm{lc}(f))\, p - \mathrm{lc}(p)/\gcd(\mathrm{lc}(p),\mathrm{lc}(f))\, f \times \mathrm{lm}(p)/\mathrm{lm}(f) \tag{2}$$

$$p \underset{f}{\longrightarrow} g = p - \lfloor \mathrm{lc}(p)/\mathrm{lc}(f) \rfloor\, f \times \mathrm{lm}(p)/\mathrm{lm}(f) \tag{3}$$

При первом способе [13] разрешается умножать приводимый многочлен на коэффициент так, чтобы стало возможно приведение в обычном смысле (1). Этот прием использует "псевдоделение". Соответственно, при такой организации редукций многочлены можно сокращать на их содержание, чтобы уменьшить рост коэффициентов. Для первого способа приведения важно эффективно реализовать алгоритм наибольшего общего делителя. С этой целью отдельно выделяются три случая параметрического задания коэффициентов: от одного, двух и $n \geq 3$ – параметров.

При втором способе запрещается домножать приводимый многочлен. Частное от деления с остатком в кольце обозначено через $\lfloor\ \rfloor$. Это прием носит название "работа над кольцом" и интересен для многих алгебраических задач и для определения соотношений на параметры коэффициентов. Второй способ гораздо более трудоемкий относительно и времени вычислений и требуемой памяти, что связанно с увеличением числа элементарных приведений (редукций) многочленов и ростом коэффициентов, поскольку нельзя сокращать многочлены на их содержание.

## 5.1 Интерфейс

**class `CoeffInterface`**(*type, systemType, parametr=None, modularShort=None*)

Задает правила работы с коэффициентом

**`type`:** строка определяющая тип коэффициента

| **Значение type** | **Тип коэффициента** |
|---|---|
| `"GmpQ"` | рациональные числа $\mathbb{Q}$ на основе библиотеки *gmp* |
| `"GmpZ"` | целые числа $\mathbb{Z}$ на основе библиотеки *gmp* |
| `"GmpZRing"` | целые числа $\mathbb{Z}$ на основе библиотеки *gmp*, работа над кольцом (3) |
| `"ModularShort"` | числа по модулю простого числа $p$ при условии $p^2 <$ машинного слова |
| `"OneParametrModularShort"` | многочлены от одной переменной с коэффициентами по модулю простого числа $p$ при условии $p^2 <$ машинного слова |
| `"TwoParametrModularShort"` | многочлены от двух переменных с коэффициентами по модулю простого числа $p$ при условии $p^2 <$ машинного слова |
| `"OneParametrGmpZ"` | многочлены от одной переменной коэффициентами из $\mathbb{Z}$ на основе библиотеки *gmp* |
| `"TwoParametrGmpZ"` | многочлены от двух переменных с коэффициентами из $\mathbb{Z}$ на основе библиотеки *gmp* |
| `"AlgebraicFieldExtGmpQ"` | расширение алгебраического поля задаваемое многочленом от одной переменной с коэффициентами из $\mathbb{Q}$ |
| `"AlgebraicFieldExtModularShort"` | расширение алгебраического поля задаваемое многочленом от одной переменной с коэффициентами по модулю простого числа $p$ при условии $p^2 <$ машинного слова |
| `"AlgebraicFieldExtNParameter"` | расширение алгебраического поля задаваемое многочленом от нескольких переменных с коэффициентами из $\mathbb{Q}$ |
| `"NParameterGmpZ"` | многочлены от нескольких переменных коэффициентами из $\mathbb{Z}$ на основе библиотеки *gmp* |
| `"NParameterField"` | рациональная функция от нескольких переменных коэффициентами из $\mathbb{Z}$ на основе библиотеки *gmp* |

**systemType:** определяет тип системы (см. 3)

**parametr:** список параметров, имеет смысл только для коэффициентов с параметрами

**modularShort:** простое число $p$ при условии $p^2 <$ машинного слова, имеет смысл для коэффициентов с модулярной арифметикой

**extension:** задает многочлен для расширения алгебраического поля

Класс `CoeffInterface` имеет следующие методы:

**type**()
: возвращает строку определяющую тип коэффициента

**isField**()
: возвращает `True` если коэффициенты многочлена образуют поле, иначе `False`

**isPseudo**()
: возвращает `True` если коэффициенты многочлена образуют кольцо, иначе `False`

**parametr**()
: возвращает список параметров

## 5.2 Коэффициент

**class Coeff**(*coeffInterface, coeff='0'*)
Задает коэффициент

**coeffInterface:** определяет интерфейс коэффициента (см. 5.1)

**coeff:** строка инициализирующая коэффициент

Класс `Coeff` имеет следующие методы:

**isZero**()
возвращает `True` если коэффициент равен нулю, иначе `False`

**isOne**()
возвращает `True` если коэффициент равен единице, иначе `False`

**setZero**()
делает коэффициент равным нулю

**setOne**()
делает коэффициент равным единице

**gcd**(*coeff*)
возвращает НОД коэффициента и коэффициента *coeff*.

Коэффициенты должны иметь одинаковый интерфейс.

**diff**(*par, deg=1*)
возвращает производную коэффициента по параметру номером *par* в степени *deg*.

Класс `Coeff` может быть аргументом следующих функций:

**str**(*coeff*)
возвращает представление в виде строки коэффициента *coeff* согласно типу системы 3.

Аналогично работает команда Python **print**

**+, -, *, /**(*coeff1, coeff2*)
Эти операции возвращают сумму, разность, произведение и частное коэффициентов. Для выполнения деление *coeff1* должен делится на *coeff2*.

Коэффициенты должны иметь одинаковый интерфейс.

**-**(*coeff*)
возвращает отрицательный коэффициент.

**+=, -=, *=, /=**(*coeff1, coeff2*)
Эти операции присваивают сумму, разность, произведение и частное коэффициентов. Для выполнения деление *coeff1* должен нацело делится на *coeff2*.

Коэффициенты должны иметь одинаковый интерфейс.

Коэффициент может быть использован в логических выражениях. Нулевой коэффициент дает в логических выражениях `False`, а остальные `True`.

Небольшая программа вычисления НОД от нескольких переменных (пример взят из работы [14]):

```
import time, ginv

st = ginv.SystemType("Polynomial")
ic = ginv.CoeffInterface("NParameterGmpZ", st, ['t','x','y','z'])
im = ginv.MonomInterface("DegRevLex", st, ['a'])
ip = ginv.PolyInterface("PolyList", st, im, ic)
```

```
g = ginv.Poly(ip, """(10 - 4*t)*x^2 - 5*x*z^2 + (4*t + 1)*x*z
  + (11 - 17*t^2 + 9*t)*x- 19*z^2 + (-7*t + 6)*z + (-11*t^2 + 15*t + 3)""").lc()
a = ginv.Poly(ip, """(18 + 10*t)*x^2 + 10*x*z^2 + (17*t + 2)*x*z
  + (2 + 17*t^2 + 8*t)*x +6*z^2 + (17*t + 6)*z + (4*t^2 - 4*t + 2)""").lc()
b = ginv.Poly(ip, """(-8-11*t)*x*2 -14*x*z^2 +(8*t-4)*x*z
  +(-17-5*t^2 +19*t)*x-11*z^2 + (17*t - 4)*z + (-14*t^2 - 19*t -2)""").lc()

def power(a, n):
  assert n >= 0
  if n == 0:
    return ginv.Coeff(ic, "1")
  else:
    return reduce(lambda x, y: x*y, [a for i in xrange(0, n)])

n = 10
for k in xrange(0, n+1):
  f1 = power(g, k)*power(a, n-k)
  f2 = power(g, k)*power(b, n-k)
  c = time.clock()
  f = f1.gcd(f2)
  print n, n-k, time.clock()-c
```

В результате будет распечатан следующий результат, в последней колонке время расчета в секундах:

```
10 10 0.12
10 9 0.32
10 8 0.35
10 7 0.58
10 6 0.94
10 5 1.75
10 4 5.83
10 3 13.63
10 2 45.08
10 1 105.11
10 0 0.0
```

# 6 Полиномы

В `ginv` реализованы следующие представления полиномов:

**Полином поддерживает двунаправленные итераторы:** `"PolyArray"` (в реализации)

**Полином не поддерживает двунаправленные итераторы:** `"PolyList"`

## 6.1 Интерфейс

**class `PolyInterface`**(*type, systemType, monomInterface, coeffInterface*)
Задает правила работы с полиномами

**`type`:** строка определяющая представление полинома

| Значение `type` | Представление полинома |
|---|---|
| `"PolyList"` | Представляет собой односвязный список членов полинома |
| `"PolyArray"` | Представляет собой массив указателей на члены полинома |

**`systemType`:** определяет тип системы (см. 3)

**`monomInterface`:** определяет интерфейс мономов полинома (см. 4.1)

**`coeffInterface`:** определяет интерфейс коэффициентов полинома (см. 5.1)

Класс `PolyInterface` имеет следующие методы:

**`type`**()
возвращает строку определяющую представление полинома

## 6.2 Полином

**class `Poly`**(*polyInterface, poly='0'*)
Задает полином

**`polyInterface`:** определяет интерфейс полинома (см. 6.1)

**`poly`:** строка инициализирующая полином (см. 3)

Класс `Poly` имеет следующие методы:

**`length`**()
возвращает количество членов полинома

**`degree`**()
возвращает максимальную сумму степеней переменных монома.

Аналогично `degree` класса `Monom`

**`length`**()
возвращает количество членов полинома

**`norm`**()
возвращает норму полинома

**`lm`**()
возвращает лидирующий моном полинома

**lc**()
: возвращает лидирующий коэффициент полинома

**setZero**()
: делает полином равным нулю

**pp**()
: делит многочлен на его содержание

**isPp**()
: возвращает `True` если содержание полинома равно 1, иначе `False`

**prolong**(*var, deg=1*)
: умножает полином на переменную с номером *var* в степени *deg*

**mult**(*coeff*)
: умножает полином на коэффциент

  Коэффициенты полинома и *coeff* должны иметь одинаковый интерфейс.

**mult**(*monom*)
: умножает полином на моном

  Мономы полинома и *monom* должны иметь одинаковый интерфейс.

**diff**(*par, deg=1*)
: возвращает производную полинома по параметру номером *par* в степени *deg*.

**reduction**(*poly*)
: выполняет приведение полинома по полиному *poly*.

  Лидирующий моном *poly* должен делит лидирующий моном полинома. Полиномы должны иметь одинаковый интерфейс.

**spoly**(*poly*)
: строит *S*-полином полинома и полинома *poly*.

  Полиномы должны иметь одинаковый интерфейс.

Класс `Poly` может быть аргументом следующих функций:

**str**(*poly*)
: возвращает представление в виде строки полинома *poly* согласно типу системы 3.

  Аналогично работает команда Python **print**

**cmp**(*poly1, poly2*)
: возвращает `1` если `poly1 > poly2`,
  возвращает `0` если `poly1 == poly2`,
  возвращает `-1` если `poly1 < poly2`.

  В сравнении участвуют лидирующие мономы полиномов. Полиномы должны иметь одинаковый интерфейс.

**<, >, <=, >=, ==, !=**(*poly1, poly2*)
: Для этих операций лидирующие мономы полиномов сравниваются согласно определенному типу упорядочения. Полиномы должны иметь одинаковый интерфейс.

  Полином может быть использован в логических выражениях. Нулевой полином дает в логических выражениях `False`, а остальные `True`

**+, /, ***(*poly1, poly2*)
: Эти операции возвращают сумму, разность и произведение полиномов.

  Полиномы должны иметь одинаковый интерфейс.

**+=, /=, *=**(*poly1, poly2*)
: Эти операции присваивает сумму, разность и произведение полиномов переменной *poly1*.

Полиномы должны иметь одинаковый интерфейс.

**len**(*poly*)
возвращает число членов полинома *poly*.

**[]**(*poly, i*)
возвращает *i*-ый член полинома.

Полином представляет собой итератор языка Python:

```
import ginv

st = ginv.SystemType("Polynomial")
im = ginv.MonomInterface("Lex", st, ['x', 'y', 'z'])
ic = ginv.CoeffInterface("GmpZ", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)

poly1 = ginv.Poly(ip, "(y^3 - x)^3")
print poly1
poly2 = ginv.Poly(ip, "(y^3 - x)^2 + (x^3 - y)^2")
print poly2
poly1 *= poly2

for (m, c) in poly1: print (m, c),
```

В результате будет распечатан следующий результат:

```
(-1)*x^3 + 3*x^2*y^3 + (-3)*x*y^6 + y^9
x^6 + (-2)*x^3*y + x^2 + (-2)*x*y^3 + y^6 + y^2
(x^9, -1) (x^8*y^3, 3) (x^7*y^6, -3) (x^6*y^9, 1) (x^6*y, 2) (x^5*y^4, -6)
 (x^5, -1)
(x^4*y^7, 6) (x^4*y^3, 5) (x^3*y^10, -2) (x^3*y^6, -10) (x^3*y^2, -1)
 (x^2*y^9, 10)
(x^2*y^5, 3) (x*y^12, -5) (x*y^8, -3) (y^15, 1) (y^11, 1)
```

# 7 Критерии

В `ginv` реализованы следующие критерии равенства нулю $S$-полиномов:

**Без критериев:** `"Without"`

**Частичные критерии Бухбергера:** `"C1"`, `"CritPartially"`

**Полные критерии Бухбергера:** `"C1C2C3"`, `"C1C2C3C4"`

**Note:** Для модульных упорядочений работает только цепочный (второй) критерий Бухбергера.

## 7.1 Интерфейс

**class `WrapInterface`**(*type, polyInterface*)
Задает правила работы с критериями

**`type`:** строка определяющая вид критерия

| Значение `type` | Вид критерия |
|---|---|
| `"Without"` | Без критериев |
| `"C1"` | Первый частичный критерии Бухбергера [6] |
| `"CritPartially"` | Первый и второй частичные критерий Бухбергера [6] |
| `"C1C2C3"` | Критерии Бухбергера [1, 2] |
| `"C1C2C3C4"` | Все критерии Бухбергера [1, 2] |

**`polyInterface`:** определяет интерфейс полинома (см. 6.1)

Класс `WrapInterface` имеет следующие методы:

**`type`**()
возвращает строку определяющую вид критерия

## 7.2 Обертка

**class `Wrap`**(*wrapInterface, poly*)
Задает обертку для организации работы критериев

**`wrapInterface`:** определяет интерфейс обертки(см. 6.1)

**`poly`:** полином (см. 6.2)

Класс `Wrap` имеет следующие методы:

**`lm`**()
возвращает лидирующий моном полинома

**`lm`**()
возвращает лидирующий моном полинома

**`ancestor`**()
возвращает лидирующий моном полинома из которого данный полином получен серией продолжений

**`poly`**()
возвращает полином

**multi**()
: возвращает количество мультипликативных переменных

**degProlong**()
: возвращает степень переменной по которой полином был получен продолжением из другого полинома

**lm**()
: возвращает лидирующий моном полинома

**isProlong**()
: возвращает `True` если является ли полином продолжением другого полнома, иначе `False`

**buildProlong**()
: возвращает список степеней переменных по которым данный полином был продолжен

Небольшая программа работы с обертками полиномов после построения базисов Грёбнера:

```
import ginv

st = ginv.SystemType("Polynomial")
im = ginv.MonomInterface("DegRevLex", st, ['x', 'y', 'z'])
ic = ginv.CoeffInterface("GmpZ", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
iD = ginv.DivisionInterface("Janet", iw)

def printWrap(w):
  print "          lm =", w.lm()
  print "    ancestor =", w.ancestor()
  print "        poly =", w.poly()
  print "       multi =", w.multi()
  print "  degProlong =", w.degProlong()
  print "   isProlong =", w.isProlong()
  print "buildProlong =", w.buildProlong()

basis = ginv.basisBuild("TQBlockLow", iD,\
  ['x^3 - y^2 + z - 1',\
   'y^3 - z^2 + x - 1',\
   'z^3 - x^2 + y - 1'])

for i in [0, 6, 12]:
  printWrap(basis[i])
```

В результате будет распечатан следующий результат:

```
          lm = x^2*y^2*z^3
    ancestor = z^3
        poly = x^2*y^2*z^3 + (-1)*x^2*y^2 + x*y^2*z + x^2*z^2 +
(-1)*x*y*z^2 + x^2*y + (-1)*x*y^2 + x^2 + (-1)*x*y + (-1)*y^2 + z + (-1)
       multi = 1
  degProlong = 0
   isProlong = True
buildProlong = 1, 1, 0
          lm = x^2*y^3
    ancestor = y^3
        poly = x^2*y^3 + (-1)*x^2*z^2 + (-1)*x^2 + y^2 + (-1)*z + 1
```

```
       multi = 2
  degProlong = 0
   isProlong = True
buildProlong = 1, 0, 0
          lm = z^3
    ancestor = z^3
        poly = z^3 + (-1)*x^2 + y + (-1)
       multi = 1
  degProlong = 0
   isProlong = False
buildProlong = 1, 1, 0
```

# 8 Инволютивные деления

В `ginv` реализованы следующие виды инволютивных делений:

**Основаны на дереве *Janet*:** `"Janet"`, `"JanetLike"`

**Note:** Дерево *Janet* позволяет за время $O(d)$, где $d$ сумма степеней монома, находить инволютивного делителя

## 8.1 Интерфейс

**class `DivisionInterface`**(*type, wrapInterface*)
Задает правила работы с инволютивным делением

**`type`:** строка определяющая тип инволютивного деления

| **Значение `type`** | **Тип инволютивного деления** |
|---|---|
| `"Janet"` | деление *Janet* [11] |
| `"JanetLike"` | деление *JanetLike* [8], очень похоже по структуре на *Janet*, но может гораздо более компактное представление на ненульмерных идеалах |

**`wrapInterface`:** определяет вид применяемых критериев (см. 7.1)

Класс `DivisionInterface` имеет следующие методы:

**`type`**()
возвращает строку определяющую тип инволютивного деления

# 9 Алгоритмы

Для построения базиса системы полиномов используется следующая функция:

**basis**(*algorithm, divisionInterface, system*)
возвращает объект класса `Basis` представляющий инволютивный базис системы *system*

**algorithm:** строка определяющая используемый алгоритм

| Значение `algorithm` | Используемый алгоритм |
|---|---|
| `"TQ"` | алгоритм TQ [7] работает на всех видах упорядочения |
| `"TQDegree"` | алгоритм TQDegree [12] работает на упорядочениях, совместимых с полной степенью (см. 4) |
| `"TQBlockHigh"` | алгоритм TQBlockHigh [9, 10] работает на упорядочениях, совместимых с полной степенью (см. 4) |
| `"TQBlockLow"` | алгоритм TQBlockLow [9, 10] работает на упорядочениях, совместимых с полной степенью (см. 4) |

**divisionInterface:** определяет интерфейс инволютивного деления (см. 8.1) используемого при построении базиса

**system:** список представляющий собой полиномы представленные объектами класса `Poly` или строк их инициализации (см. 6.2)

## 9.1 Basis

**class Basis**
Представляет собой инволютивный базис. Может быть сконструирован только функцией `basis`.

Класс `Basis` имеет следующие методы:

**lenghtIB**()
возвращает количество элементов инволютивного базиса

**lenghtGB**()
возвращает количество элементов базиса Грёбнера

**numCriterion**()
возвращает количество примененных критериев равенства нулю $S$-полиномов при построении базиса

**numSpoly**()
возвращает количество вычисленных $S$-полиномов при построении базиса

**numReduction**()
возвращает количество выполненных приведений полиномов при построении базиса

**userTime**()
возвращает затраченное *пользовательское время* при построении базиса

**sysTime**()
возвращает затраченное *системное время* при построении базиса

**realTime**()
возвращает затраченное *астрономическое время* при построении базиса

**hilbertPolynomial**()
возвращает строку представляющую полином Гильберта инволютивного базиса

**iterStatistics()**
возвращает итератор для просмотра статистики работы инволютивного алгоритма. Представляет собой кортеж из 4 элементов:

- время шага
- количество элементов в множестве $T$
- количество элементов в множестве $Q$
- количество элементов в множестве $Q$ имеющих минимальный лидирующий мономом

**iterIB()**
возвращает итератор для просмотра инволютивного базиса представленного объектами класса `Poly` (см. 6.2)

**iterGB()**
возвращает итератор для просмотра базиса Грёбнера представленного обьектами класса `Poly` (см. 6.2)

**iterLmIB()**
возвращает итератор для просмотра лидирующих мономов инволютивного базиса представленного объектами класса `Monom` (см. 4.2)

**iterLmGB()**
возвращает итератор для просмотра лидирующих мономов базиса Грёбнера представленного объектами класса `Monom` (см. 4.2)

**iterLmGB()**
возвращает итератор для просмотра лидирующих мономов базиса Грёбнера представленного объектами класса `Monom` (см. 4.2)

**find**(*monom*)
ищет инволютивный делитель в базисе. *monom* может представлять собой строку инициализации монома или представлять собой объект класса `Monom` (см. 4.2).

Мономы в базисе и *monom* должны иметь одинаковые интерфейсы.

**nf**(*poly*)
выполняет приведение *poly* по базису с точностью до лидирующего монома. *poly* может представлять собой строку инициализации монома или представлять собой объект класса `Poly` (см. 6.2).

Полиномы в базисе и *poly* должны иметь одинаковые интерфейсы.

**isNf**(*poly*)
выполняет проверку приведения *poly* по базису с точностью до лидирующего монома. Возвращает `True`, иначе `False` *poly* может представлять собой строку инициализации монома или представлять собой объект класса `Poly` (см. 6.2).

Полиномы в базисе и *poly* должны иметь одинаковые интерфейсы.

**nfTail**(*poly*)
выполняет полное приведение *poly* по базису без редукции лидирующего монома. *poly* может представлять собой строку инициализации монома или представлять собой объект класса `Poly` (см. 6.2).

Полиномы в базисе и *poly* должны иметь одинаковые интерфейсы.

**isNf**(*poly*)
выполняет проверку приведения *poly*по базису без редукции лидирующего монома. Возвращает `True`, иначе `False` *poly* может представлять собой строку инициализации монома или представлять собой объект класса `Poly` (см. 6.2).

Полиномы в базисе и *poly* должны иметь одинаковые интерфейсы.

Класс `Basis` может быть аргументом следующих функций:

**len**(*basis*)
возвращает число элементов инволютивного базиса *basis*.

**[]**(*basis, i*)
возвращает *i*-ый член инволютивного базиса представленный объектами класса `Wrap` (см. 7.2)

Базис представляет собой итератор языка Python:

```
import ginv

st = ginv.SystemType("Polynomial")
im = ginv.MonomInterface("Lex", st, ['x', 'y', 'z'])
ic = ginv.CoeffInterface("GmpZ", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
iD = ginv.DivisionInterface("Janet", iw)

basis = ginv.basisBuild("TQ", iD,\
  ['x^3 - y^2 + z - 1',\
   'y^3 - z^2 + x - 1',\
   'z^3 - x^2 + y - 1'])

for w in basis:
  print w.lm(),
print
```

В результате будет распечатан список лидирующих мономов инволютивного базиса '`x y z^27`'.

# Список литературы


[1] J. Apel and R. Hemmecke. *Detecting Unnecessary Reductions in an Involutive Basis Computation*. RISC Report 02-22, Linz, 2002.

[2] J. Apel and R. Hemmecke. Detecting unnecessary reductions in an involutive basis computation. *Journal of Symbolic Computation*, 40(4-5):1131–1149, 2005.

[3] Yu. A. Blinkov, C. F. Cid, V. P. Gerdt, W. Plesken, and D. Robertz. The MAPLE package "Janet": I. Polynomial systems. In V.G. Ganzha, E.W. Mayr, and E.V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, pages 31–40. Institut für Informatik, Technische Universität München, Garching, 2003.

[4] Yu. A. Blinkov and V. P. Gerdt. Specialized computer algebra system GINV. *Programming and Computer Software*, 34(2):112–123, 2008.

[5] W. C. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *Journal of the ACM*, 18(4):476–504, 1971.

[6] V. P. Gerdt and Yu. A. Blinkov. Involutive bases of polynomial ideals. *Mathematics and Computers in Simulation*, 45:519–542, 1998.

[7] V. P. Gerdt and Yu. A. Blinkov. Minimal involutive bases. *Mathematics and Computers in Simulation*, 45:543–560, 1998.

[8] V. P. Gerdt and Yu. A. Blinkov. Janet-like monomial division. In *Computer Algebra in Scientific Computing*, volume 3718 of *Lecture Notes in Computer Science*, pages 174–183. Springer Berlin / Heidelberg, 2005.

[9] V. P. Gerdt and Yu. A. Blinkov. On computing Janet bases for degree compatible orderings. In J. Draisma and H. Kraft, editors, *Proceedings of the Rhein Workshop on Computer Algebra*, pages 107–117. University of Basel, 2006. math.AC/0603161.

[10] V. P. Gerdt and Yu. A. Blinkov. On selection of nonmultiplicative prolongations in computation of Janet bases. *Programming and Computer Software*, 33(3):147–153, 2007.

[11] V. P. Gerdt, D. A. Yanovich, and Yu. A. Blinkov. Construction of Janet bases I. Monomial bases. In V.G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, Lecture Notes in Computer Science, pages 233–247. Springer-Verlag Berlin, 2001.

[12] V. P. Gerdt, D. A. Yanovich, and Yu. A. Blinkov. Construction of Janet bases II. Polynomial bases. In V.G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, Lecture Notes in Computer Science, pages 249–265. Springer-Verlag Berlin, 2001.

[13] A. Kandri-Rody and D. Kapur. Computing the Gröbner-basis of an Ideal in Polynomial Rings over the Integers. In *Proceedings 1984 MACSYMA User's Conference*, pages 436–451, 1984.

[14] Mark van Hoeij and Michael B. Monagan. Algorithms for polynomial gcd computation over algebraic function fields. In Jaime Gutierrez, editor, *ISSAC*, pages 297–304. ACM, 2004.

[15] В. П. Гердт and Ю. А. Блинков. О стратегии выбора немультипликативных продолжений при вычислении базисов Жане. *Программирование*, 33(3):34–43, 2007.

# A Примеры

Примеры содержатся в папке 'examples'

## A.1 Полиномы

Для запуска полиномиального примера достаточно открыть в **idle** 'examples.py' или набрать в папке примеров командную строку

```
python example.py
```

Выбор примера можно осуществить из следующего списка:

```
...
# assur44       ducos7_3   hf855           quadfor2
# aubry2        ducos7_5   hietarinta1     quadgrid
# augot         ducos8     hunecke         rabmo
# benchmark_D1  eco10      i1              rbpl24
# benchmark_i1  eco11      ilias12         rbpl
# boon          eco12      ilias13         redcyc5
# butcher8      eco7       ilias_k_2       redcyc6
# butcher       eco8       ilias_k_3       redcyc7
# camera1s      eco9       issac97         redcyc8
# caprasse      el44       jcf26           redeco10
# cassou        el50       katsura10       redeco11
# chandra4      extcyc4    katsura6        redeco12
# chandra5      extcyc5    katsura7        redeco7
# chandra6      extcyc6    katsura8        redeco8
# chemequs      extcyc7    katsura9        redeco9
# chemequ       extcyc8    kin1            reif
# chemkin       f633       kinema          reimer4
# cohn2         f744       kotsireas       reimer5
# cohn3         f855       ku10            reimer6
# comb3000s     f966       lanconelli      reimer7
# comb3000      fabrice24  lichtblau       reimer8
# conform1      filter9    liu             rose
# cpdm5         geneig     lorentz         s9_1
# cyclic5       hairer1    matrix          solotarev
# cyclic6       hairer2    mckay.gls50mod  sparse5
# cyclic7       hairer3    mckay           speer
# cyclic8       hairer4    mickey          tangents
# d1            hawes4     morgenstern     test
# des18_3       hcyclic5   noon5           uteshev_bikker
# des22_24      hcyclic6   noon6           vermeer
# dessin1       hcyclic7   noon7           vernov1
# dessin2       hcyclic8   noon8           virasoro
# discret3      heart      noon9           wang16
# dl            hemmecke   pinchon1        wright
# ducos10       hf744      puma

folder = os.path.join(".", "polynomial")
fname = "cyclic7" + ".xml.gz"
...
```

отредактировав в нижней строке имя файла. Его описание выводится на экран командой

```
print description
```

Также узнать о примере можно просмотрев его во внутренней папке ‘polynomial’. Примеры хранятся в *xml*-файлах сжатых архиватором *gzip*.

Объединение идеалов можно построить следующей программой:

```
import ginv

st = ginv.SystemType("Polynomial")
im = ginv.MonomInterface("DegRevLex", st, ["x"])
ic = ginv.CoeffInterface("GmpZ", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
id = ginv.DivisionInterface("JanetLike", iw)

I = ["(x^3-1)*(x^2+2*x)"]
J = ["(x^3+1)*(x^2+2*x)"]
IJ = I + J
print IJ

basis = ginv.basisBuild("TQ", id, IJ)
for p in basis.iterGB():
  print p
```

Поскольку `I` и `J` являются идеалами, порождаемые многочленами от одной переменной, то приведенный базис Грёбнера их объединения является их наибольшим общим делителем.

```
['(x^3-1)*(x^2+2*x)', '(x^3+1)*(x^2+2*x)']
x^2 + 2*x
```

Для конструктивного определения принадлежности радикалу идеала нужно ввести дополнительную переменную `@` и добавить дополнительное уравнение:

```
import ginv

st = ginv.SystemType("Polynomial")
im = ginv.MonomInterface("DegRevLex", st, ["x", "y", "@"])
ic = ginv.CoeffInterface("GmpZ", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
id = ginv.DivisionInterface("JanetLike", iw)

I = ["(x-y)^4*(x^2+y^2)^3"]
eq = "(x-y)*(x^2+y^2)"
I.append("1-@*%s" % eq)
print I

basis = ginv.basisBuild("TQ", id, I)
if basis[0].lm().degree() == 0:
  print "Belongs"
else:
  print "Does not belong"
```

Условие `basis[0].lm().degree() == 0` показывает, что система несовместна и уравнение `eq` принадлежит идеалу.

```
['(x-y)^4*(x^2+y^2)^3', '1-@*(x-y)*(x^2+y^2)']
Belongs
```

## A.2 Модули

Приведение по способу (3) демонстрирует следующая программа:

```
import ginv

st = ginv.SystemType("Polynomial", module=3)
im = ginv.MonomInterface("TopDegRevLex", st, ['x', 'y'])
ic = ginv.CoeffInterface("GmpZ" + "Ring", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
id = ginv.DivisionInterface("Janet", iw)

equations = ['[-25*y, 260*y, (-130*x-195)*y]',
  '[(-2*x^2+2)*y, (-19+20*x^2)*y, (-10*x^3+10*x-15*x^2+15)*y]',
  '[(-4*x+6)*y, (40*x-60)*y, (46-20*x^2)*y]']

basis = ginv.basisBuild("TQ", id, equations)

for p in basis.iterIB():
  print p

print "nf =", basis.nf("[6*y, -51*y, (26*x + 39)*y]")
```

Результатом работы будет следующий базис:

```
[0, (-2)*x*y + 3*y, x^2*y + (-1)*y]
[x*y + y, (-26)*y, 13*x*y + 13*y]
[5*y, (-52)*y, 26*x*y + 39*y]
nf = [y, y, 0]
```

Значение функции `nf` показывает, что элемент `[6*y, -51*y, (26*x + 39)*y]` не принадлежит идеалу.

Если базис считать по первому способу (2), в ответе получим более простой базис:

```
[2*x*y + (-3)*y, 0, (-13)*y]
[5*y, (-52)*y, 26*x*y + 39*y]
nf = [y, y, 0]
```

Работу в алгебраических расширениях демонстрирует следующая программа:

```
import ginv

st = ginv.SystemType("Polynomial", module=3)
im = ginv.MonomInterface("TopDegRevLex", st, ['x', 'y'])
ic = ginv.CoeffInterface("AlgebraicFieldExtGmpQ", st,  parameter=['a'],
```

```
  extension=['(a+1)^3-a^2'])
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
id = ginv.DivisionInterface("Janet", iw)

equations = ['[-25*a^2*y, 26*y + 12/3*a, (-13*x-19-a)*y]',
  '[(-2*x^2+2*a)*y, (-19+20/7*x^2)*y + 13*a, (-10*x^3*a+10*x-15*x^2+15*a)*y]']

basis = ginv.basisBuild("TQ", id, equations)

for p in basis.iterIB():
  print p
```

```
[x^2*y + (228500/4244947*a^2 - 72525125/110368622*a -
133749750/55184311)*x*y + (563028125/1434792086*a^2 +
3271775289/1434792086*a + 3733278625/717396043)*y,
(-54973750/29714629*a^2 - 95607460/29714629*a -
127405720/29714629)*x^2*y + (185880/4244947*a^2 +
592500/4244947*a + 905000/4244947)*x^2 + (29198935/4244947*a^2 +
47697890/4244947*a + 65959205/4244947)*x*y +
(-21399960/55184311*a^2 - 43275200/55184311*a -
58397870/55184311)*x + (-1106094695/110368622*a^2 -
69133370/4244947*a - 2488133481/110368622)*y +
(1206212145/717396043*a^2 + 2250565353/717396043*a +
5924760265/1434792086)*1, (2136627665/204970298*a^2 +
3471555155/204970298*a + 2403279155/102485149)*y]
[25/13*a^2*y, (-2)*y + (-4/13*a)*1, x*y + (1/13*a + 19/13)*y]
```

Следующий пример демонстрирует вычисление с параметрами первым способом (2):

```
import ginv

st = ginv.SystemType("Polynomial")
ic = ginv.CoeffInterface("NParameterGmpZ", st, ['c[1]','c[2]','c[8]'])
im = ginv.MonomInterface("DegRevLex", st, ['a[1]', 'a[3]', 'b[1]'])
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("Without", ip)
iD = ginv.DivisionInterface("Janet", iw)

eqs = ["(-a[1])*b[1] + (-c[1])",\
  "(2) + (-1)*b[1]^2 + (-1)*a[1]^2 + (-c[2])",\
  "(-1)*a[3]^2 + (-c[8])"]

basis = ginv.basisBuild("TQ", iD, eqs)

for p in basis.iterIB():
  print p
```

```
a[3]*b[1]^3 + (-1*c[1])*a[1]*a[3] + (c[2] - 2)*a[3]*b[1]
a[1]*a[3]^2 + c[8]*a[1]
a[1]*a[3]*b[1] + c[1]*a[3]
b[1]^3 + (-1*c[1])*a[1] + (c[2] - 2)*b[1]
a[1]^2 + b[1]^2 + (c[2] - 2)*1
```

```
a[3]^2 + c[8]*1
a[1]*b[1] + c[1]*1
```

Построение пересечения идеалов можно осуществить задав полиномиальный модуль и выбрав упорядочение, исключающие зависисимые переменные:

```
import ginv

st = ginv.SystemType("Polynomial", module=2)
im = ginv.MonomInterface("PosElimLex", st, ["x", "y"], tupSep=1)
ic = ginv.CoeffInterface("GmpZ", st)
ip = ginv.PolyInterface("PolyList", st, im, ic)
iw = ginv.WrapInterface("CritPartially", ip)
id = ginv.DivisionInterface("JanetLike", iw)

I = ["(x-y)*(x+y)"]
J = ["(x+y)*(x+y)"]
IJ = ["[%s, 0]" % p for p in I] + ["[%s, %s]" % (p, p) for p in J]
print IJ

basis = ginv.basisBuild("TQ", id, IJ)
for p in basis.iterGB():
  if str(p)[1] == '0':
    print str(p)[4:-1]
```

В результате, приведенный базис Грёбнера пересечения идеалов состоит из элементов, содержащих 0 в качестве первой компоненты. Поскольку `I` и `J` - главные идеалы, то приведенный базис Грёбнера их пересечения содержит наименьшее общее кратное их образующих.

```
['[(x-y)*(x+y), 0]', '[(x+y)*(x+y), (x+y)*(x+y)]']
x^3 + x^2*y + (-1)*x*y^2 + (-1)*y^3
```

## A.3 Дифференциальные уравнения

## A.4 Разностные уравнения

# B Сравнение с другими программами

В этом разделе представлены сравнение *GINV* с *JB* и *Magma* (см. [9]http://arXiv.org/math.AC/0603161 для более подробной информации).

Приведенные в таблице времена были получены на следующих компьютерах:

**JB:** 2xOpteron-242 (1.6 Ггц) с 4Гб памяти, работающий под Gentoo Linux 2004.3 и с компиллятором gcc-3.4.2.

**GINV:** Turion-3400 (1.8 Ггц) с 2Гб памяти, работающий под Gentoo Linux 2005.1 с компиллятором gcc-3.4.4.

**Magma:** дуальный процессор Pentium III (1 ГГц) с 2 Гб памяти, работающий под SuSE Linux 8.0 (ядро 2.4.18-64GB-SMP) с компиллятором gcc-2.95.3.

Все времена указаны в секундах. При этом для примеров отмеченных (*) базис Грёбнера подсчитать не удалось из-за переполнения памяти компьютера.

Таблица 1: Времена счета

| Пример | JB | GINV | | | Magma | |
|---|---|---|---|---|---|---|
| | TQDegree | TQDegree | TQBlockHigh | TQBlockLow | V2.11-8 | V2.12-17 |
| assur44 | 10.35 | 14.20 | 6.33 | 6.4 | 4.56 | 4.99 |
| butcher8 | 1.06 | 1.02 | 0.38 | 0.39 | 4.68 | 5.00 |
| chemequs | 0.67 | 0.61 | 0.57 | 0.6 | 12.80 | 11.99 |
| chemkin | 17.83 | 16.87 | 10.95 | 9.95 | 32.34 | 29.83 |
| cohn3 | 76.72 | 107.14 | 30.21 | 25.47 | 37.73 | 39.20 |
| cpdm5 | 1.78 | 1.57 | 1.69 | 1.68 | 0.69 | 0.70 |
| cyclic6 | 0.12 | 0.19 | 0.14 | 0.14 | 0.09 | 0.08 |
| cyclic7 | 58.72 | 60.94 | 68.59 | 65.28 | 6.64 | 7.08 |
| cyclic8 | 12056.24 | 14046.26 | 5826.18 | 4424.96 | 235.73 | 245.65 |
| d1 | 8.77 | 12.58 | 1.99 | 2.08 | 28.49 | 8.29 |
| des18_3 | 0.19 | 0.18 | 0.19 | 0.19 | 1.81 | 1.89 |
| des22_24 | 0.68 | 0.62 | 0.77 | 0.79 | 1.37 | 1.46 |
| discret3 | 23322.8 | 20956.31 | 12642.49 | 13521.65 | 33658.09 | 19369.53 |
| dl | 270.17 | 278.89 | 80.77 | 89.52 | 14.57 | 11.95 |
| eco8 | 0.40 | 0.44 | 0.44 | 0.46 | 0.20 | 0.20 |
| eco9 | 3.22 | 5.60 | 4.99 | 5.08 | 1.25 | 1.20 |
| eco10 | 52.56 | 56.70 | 65.71 | 68.06 | 7.07 | 6.91 |
| eco11 | 765.98 | 741.74 | 718.53 | 679.3 | 62.33 | 51.08 |
| extcyc5 | 1.35 | 1.53 | 1.46 | 1.37 | 0.37 | 0.38 |
| extcyc6 | 324.70 | 184.49 | 276.06 | 155.64 | 45.36 | 47.96 |
| extcyc7 | * | * | * | * | 8242.00 | 8492.13 |
| f744 | 4.88 | 7.71 | 2.22 | 2.68 | 1.47 | 1.38 |

| Пример | JB | GINV | | | Magma | |
|---|---|---|---|---|---|---|
| | TQDegree | TQDegree | TQBlockHigh | TQBlockLow | V2.11-8 | V2.12-17 |
| f855 | 132.97 | 139.79 | 37.64 | 38.45 | 48.63 | 37.06 |
| fabrice24 | 108.52 | 116.77 | 8.2 | 7.7 | 9.45 | 8.70 |
| filter9 | 20.97 | 5.76 | 1.13 | 1.6 | 80.04 | 56.67 |
| hairer2 | 62.91 | 108.17 | 126.69 | 125.43 | 92.07 | 85.86 |
| hairer3 | 1.96 | 0.92 | 0.32 | 1.4 | * | * |
| hcyclic7 | 64.17 | 53.87 | 65.81 | 73.0 | 6.26 | 6.76 |
| hcyclic8 | 6024.97 | 4316.59 | * | 7560.99 | 229.70 | 237.12 |
| hf744 | 22.17 | 8.58 | 7.18 | 11.26 | 1.39 | 1.32 |
| hf855 | 2157.88 | 534.08 | 806.51 | 988.38 | 48.15 | 36.69 |
| hietarinta1 | 0.77 | 0.71 | 0.38 | 0.53 | 2.63 | 2.15 |
| i1 | 98.24 | 122.36 | 58.29 | 58.21 | 55.07 | 42.35 |
| ilias13 | 1167.18 | 5851.97 | 3013.1 | 2469.62 | 336.21 | 309.64 |
| ilias_k_2 | 323.59 | 669.68 | 445.51 | 270.21 | 55.41 | 54.71 |
| ilias_k_3 | 452.32 | 846.19 | 1162.7 | 622.14 | 90.67 | 89.97 |
| jcf26 | 224.96 | 211.24 | 16.44 | 14.65 | 31.64 | 25.59 |
| katsura7 | 2.15 | 1.77 | 2.08 | 1.98 | 0.72 | 0.79 |
| katsura8 | 27.48 | 24.66 | 28.8 | 27.09 | 4.7 | 5.06 |
| katsura9 | 337.52 | 294.59 | 340.45 | 311.98 | 33.47 | 34.87 |
| katsura10 | 4790.55 | 4983.11 | 7220.29 | 6204.95 | 287.38 | 292.02 |
| kin1 | 15.18 | 20.32 | 7.11 | 7.11 | 50.56 | 45.33 |
| kotsireas | 6.33 | 37.94 | 4.93 | 4.27 | 3.45 | 3.67 |
| noon6 | 0.97 | 1.29 | 1.27 | 1.29 | 0.60 | 0.62 |
| noon7 | 28.87 | 32.58 | 37.52 | 38.52 | 4.93 | 4.77 |
| noon8 | 1552.26 | 2292.84 | 3322.62 | 3152.57 | 43.65 | 42.80 |
| pinchon1 | 10.37 | 0.04 | 0.01 | 0.01 | 4.09 | 3.54 |
| rbpl | 210.94 | 177.51 | 173.8 | 173.98 | 38.33 | 35.79 |
| rbpl24 | 108.78 | 116.78 | 8.23 | 7.7 | 9.62 | 8.74 |
| redcyc6 | 0.16 | 0.17 | 0.13 | 0.14 | 0.10 | 0.10 |
| redcyc7 | 913.75 | 1048.69 | 48.19 | 48.61 | 5.73 | 6.36 |
| redeco10 | 18.51 | 18.66 | 23.91 | 22.4 | 2.33 | 2.40 |
| redeco11 | 178.32 | 187.36 | 253.34 | 228.41 | 14.56 | 14.85 |
| redeco12 | 1735.95 | 2172.75 | 4666.8 | 3385.97 | 101.51 | 103.02 |
| reimer5 | 0.22 | 0.36 | 0.34 | 0.38 | 0.74 | 0.70 |
| reimer6 | 9.69 | 21.60 | 24.19 | 23.96 | 42.13 | 42.40 |
| reimer7 | 719.37 | 3808.91 | 4756.4 | 4314.12 | 5216.53 | 5032.73 |
| virasoro | 9.69 | 8.90 | 10.96 | 10.68 | 1.72 | 1.77 |