pyginv User's Guide

Release 1.2

Blinkov Yu.A. and Gerdt V.P.

July 19, 2006

BlinkovUA@info.sgu.ru, gerdt@jinr.ru

Contents

---

1 Introduction

1.1 Project GINV

The open source software GINV implements the Gröbner bases method for systems of equations.

GINV is a C++ module of Python designed for constructing Gröbner bases of ideals and modules in polynomial, differential and difference rings.

Gröbner bases are constructed by involutive algorithms.

GINV is an open source software.

The source codes, the installation package for Python, documentation in Russian and English are available on the Web page http://invo.jinr.ru

1.2 Abstract

2 Quick start

2.1 Installation

Before the GINV installation one must install Python of version ¿=2.4. Python is downloadable from the Web page http://www.python.org.

To run `ginv` after its installation in a non-standard folder (for example, without the administrator rights) it is necessary to define the environmental variable PYTHONPATH. PYTHONPATH is a list of names of folders given in the same format as in the string PATH and separated by the symbol ':' (”Unix”) or ';' (”Windows”). Then the search of Python modules will be done in the indicated folders prior to their default search.

Windows

Installation under Windows uses the pre-compiled module `ginv`. This imposes some restrictions on the type and capacity of a processor. The module `ginv` is built-up with *i586* code. This may lead to a substantial slow-down in performance, for instance, when a 64-bit processor is used.

Assemblage under Windows is similar to that under Unix.

Unix

To install the GINV software under Unix one needs the library *gmp* (http://www.swox.com/gmp) assembled with *C++* support. For the code optimization under a given computer architecture one can properly adjust the *C++* compiler options in the file ‘setup.py’.

The assemblage of the module `ginv` is done in line with the standard scheme for Python:

```
python setup.py build
```

For installation into the standard folder (and assemblage if the module has not been assembled):

```
python setup.py install
```

For installation into a non-standard folder:

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

2.2 Start

To verify the installation it is sufficient to start Python and then import the module by the command

```
>>> import ginv
```

The following example allows to transform a polynomial system into a triangular form and output it on a display.

```
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
```

As a result, the three polynomials below will be displayed.

```
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 + (-310272794617140912662 27
160633475001590756076807*z^18 + 982342916167428059233558*z^17 + (-209204326958442264627 72)*z^16
123787951244996878821776 1*z^15 + (-649364739492104109294111)*z^14 + 177078379825139598960064 0*z
(-32695707856558651152676 46)*z^12 + 10171373925735485691604 25*z^11 + (-538029066871643242248011
25113184149253132713587 36*z^9 + (-58207659267432221985142 92)*z^8 + 39162341050068094253621 44*z^
(-12717409869604816493742 76)*z^6 + 45265718661283447634457 68*z^5 + 42253994071244800646475 79*z^
(-17024881399113466230548 30)*z^3 + 25993215882084627878281 65*z^2 + (-64990501983430103568775 18)
11385316410354538250716 56
139536745252397484708849 6*y + (-242632459056334226660 89)*z^26 + (-256410179757740270610 5)*z^25
(-76127169343980429525 13)*z^24 + 18370847384350866967688 88*z^23 + 27382590493064706561728*z^22 +
93382883659052502917872*z^21 + (-44501973596340773672896 5)*z^20 + (-12458032569382705296765 3)*z
(-49691351438130573128611 5)*z^18 + 68602856840796075827477 0*z^17 + (-29326453439546806834518 0)*
17549163177398696559130 83*z^15 + (-14941610444207193267581 49)*z^14 + 31668648486659224915071 68*
(-43241488667181073667198 98)*z^12 + 51181933538205943543404 67*z^11 + (-96699880419481987599255 0
82349643925309391879611 84*z^9 + (-14800290986146424015670 204)*z^8 + 14407709710372641542274 656*
(-17110238558169309305535 500)*z^6 + 19289855195567445653291 224*z^5 + (-14071675248685297091156 2
17208495316564910541540 422*z^3 + (-97802610323155958711682 33)*z^2 + 56845436070899796724432 54*z
(-46064307578045953892048 88)
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
+ (-261)*z^12 + 484*z^11 + (-545)*z^10 + 740*z^9 + (-908)*z^8 + 972*z^7 + (-1220)*z^6 + 1047*z^
(-1045)*z^4 + 943*z^3 + (-535)*z^2 + 422*z + (-216)
```

3 Input data types

The input data types determine modes for their internal representation and for operations over them. One distinguishes four different input data types: (commutative) polynomials, modules, differential and difference equations. Their principal distinction lies in the monomial representation. Commutative polynomials may have coefficients of different nature: integer/rational numbers or multi-parametric polynomials/rational functions. The input data can be represented by any admissible expressions, including fractions. A fraction must have in the denominator the same data type as in the numerator. Then, after reducing to a common denominator the last is casted out. The type of a system is given by the class `SystemType`.

class `SystemType` (*type, module=None, rightPart=None*)
Defines the type of a polynomial

`type`: string defining the type of a polynomial

Value of `type`	System type
`"Polynomial"`	Polynomial system 3.1
`"DifferentialEquation"`	System of linear differential equations 3.3
`"FiniteDifferenceScheme"`	System of linear difference equations 3.4

`module`: integer number that specifies the dimension of a module

`rightPart`: integer number that specifies the dimension of the right-hand sides for equalities, and for these sides the relations are to be obtained that characterize consistency of the system. This is applicable to modules only.

The class `SystemType` has the following attributes:

`type`
string specifying the type of a system (read-only)

`module`
integer number specifying the dimension of a module (read-only)

`rightPart`
integer number specifying the dimension of the rand-hand sides in an equality (read-only)

3.1 Polynomials

As an example, a polynomial in the variables `['x','y','z']` can be represented by a string of the form

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

In this expression `df` is the first derivative with respect to x and the second one with respect to y.

3.2 Modules

Module elements are represented by a list of polynomials

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

3.3 Differential equations

A differential equation with dependent variables `['u','v']` and independent variables `['x','y','z']` can be represented by the string

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

3.4 Difference equations

A difference equation with dependent variables `['u','v']`, dependent variables `['t','x']` and parameters `['tau','h']` can be represented by the string

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

4 Monomials

In `ginv` the following monomial orders have been implemented:

Degree compatible orders: `"TopDegRevLex"`, `"DegRevLex"`

Orders with vectorization of computations: `"TopDegRevLexByte"`, `"DegRevLexByte"`

Note: Manipulations with the machine word are the fastest ones for a processor. Vectorization allows to place several exponents in the machine word (for *C* compilers it is type *unsigned int*). Thereby, some operations over monomials are accelerated several times, and the memory consumed is also decreased. This makes sense when the exponents are accordingly bounded, and it is advisable for degree compatible orders.

For vectorization it is necessary to know where in a given computer the sign bit is located: in the beginning or in the end of the machine word. This is defined by the *C* macros *WORDS_BIGENDIAN* or *WORDS_LITTLEENDIAN*. If they are not specified, then all vectorization does is the memory optimization.

Ordesr which are not degree compatible: `"TopLex"`, `"TopElim"`, `"PotLex"`, `"PotDegRevLex"`, `"PosElim"`, `"Lex"`, `"Elim"`

4.1 Interface

class `MonomInterface`(*order, systemType, independ, depend=None, varSep=None*)

Specifies operations over monomials

`order`: string defining monomial order

Value of `order`	Order type
`"TopDegRevLex"`	”term over position” order for modules where the terms are compared, first, by their total degree and, second, reverse lexicographically
`"DegRevLex"`	total degree order where the ties are broken by the reverse lexicography
`"TopDegRevLexByte"`	”term over position” order for modules with vectorization and such that the terms are compared, first, by their total degree and, second, reverse lexicographically
`"DegRevLexByte"`	order with vectorization and such that the terms are compared, first, by their total degree and, second, reverse lexicographically
`"TopLex"`	lexicographical ”term over position” order for modules
`"TopElim"`	elimination ”term over position” order
`"PotLex"`	lexicographical ”position over term” order for modules
`"PotDegRevLex"`	”position over therm order” order for modules where the terms are compared first by their total degree and then reverse lexicographically
`"PosElim"`	elimination order for dependent variables
`"Lex"`	lexicographical order
`"Elim"`	elimination order for independent variables

`systemType`: specifies type of a system (see 3)

`independ`: list of independent variables

`depend`: list of dependent variables (currently used for modules)

`varSep`: index of separating variable (dependent or independent) used for elimination orders

The class `MonomInterface` contains the following methods:

`order``()`
returns the string specifying the order type.

`dimIndepend``()`
returns the number of independent variables.

`independ``()`
returns the list of independent variables.

`dimDepend``()`
returns the number of dependent variables.

`depend``()`
returns the list of dependent variables.

`varSep``()`
returns the index of separating variable (dependent or independent).

4.2 Monomial

class `Monom`(*monomInterface, monom=None*)
Specifies monomial

`monomInterface`: defines the interface of a monomial (see 4.1)

`monom`: string that initializes a monomial. If the string is not specified, then the monomial becomes the unit one. For module orders a Python dependent variable is not determined. It can be given by the method `setDependVar`.

Example for value of `monom`	System type
`'1'`	unit monomial for a system of any type
`'x*y^4*x'`	polynomial system 3.1
`'df(u, x, y, 4, x)'`	linear differential system 3.3
`'T(u, x, y, 4, x)'`	linear difference system 3.4

The class `Monom` contains the following methods:

`degree``()`
returns the total degree of a monomial.

`dependVar``()`
returns the index of independent variable.

`setZero``()`
zeroizes the variable degrees in a monomial.

`prolong`(*var, deg=1*)
multiplies a monomial by the independent variable with index *var* raised to power *deg*.

`gcd`(*monom*)
returns the total degree of variables in GCD of a monomial and the monomial *monom*.

The monomials must have the same interface.

`lcm`(*monom*)
returns LCM of a monomial and the monomial *monom*.

The monomials must have the same interface.

`divisibility`(*monom*)
returns `True` if a monomial is divisible by the monomial *monom*, otherwise returns `False`.

The monomials must have the same interface.

divisibilityTrue(*monom*)
returns `True` if a monomial is divisible by the monomial *monom* and they are not equal. Otherwise, returns `False`.

The monomials must have the same interface.

Class `Monom` can be an argument of the following functions:

str(*monom*)
returns the string representation of monomial *monom* in accordance to the type of system 3.

The Python command **print** works similarly

cmp(*monom1, monom2*)
returns `1` if `monom1 > monom2`,
returns `0` if `monom1 == monom2`,
returns `-1` if `monom1 < monom2`.

Monomials are compared in accordance to the order specified. The monomials must have the same interface.

<, >, <=, >=, ==, !=(*monom1, monom2*)
For these operations monomials are compared in accordance to the order specified. The monomials must have the same interface.

Monomials can be used in logical expressions. The unit monomial yields `False` in logical expressions, and other monomials yield `True`.

***, /**(*monom1, monom2*)
These operations return the product and quotient of monomials. To perform the division operation monomial *monom1* must be divisible by *monom2*.

The monomials must have the same interface.

***=**(*monom1, monom2*)
This operation assigns the product of monomials to variable *monom1*.

The monomials must have the same interface.

len(*monom*)
returns the number of independent variables in the monomial interface *monom*.

It acts similarly to method `dimIndepend` in class `MonomInterface`.

[](*monom, i*)
returns degree *i* of the independent variable in monomial *monom*.

Monomial is an iterator of 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,
```

This yields the list of degrees '`4 1 2`' printed.

5 Coefficients

In `ginv` the following types of coefficients have been implemented:

Polynomial coefficients ring: `"ITwoParametrModularShort"`, `"IOneParametrModularShort"`, `"IGmpZ"`

Polynomial coefficient field: `"IModularShort"`, `"IGmpQ"`

Note: In this case some operations over polynomials can be performed more efficiently, for example, in the modular case, if they do not need computation of GCD.

5.1 Interface

class `CoeffInterface` (*type, systemType, parametr=None, modularShort=None*)
Specifies operations over coefficients.

`type`: string that defines the type of coefficients

Value of `type`	Coefficient type
`"GmpQ"`	rational numbers $\mathbf{Q}$ on the basis of the *GMP* library
`"GmpZ"`	integer numbers $\mathbf{Z}$ on the basis of the *GMP* library
`"ModularShort"`	numbers modulo prime number p $\mathbf{Z}p$ under the condition $p^2 <$ machine word
`"OneParametrModularShort"`	polynomial arithmetics with one-parametric coefficients and integers modulo prime number p $\mathbf{Z}p$ under the condition $p^2 <$ machine word
`"TwoParametrModularShort"`	polynomial arithmetics with two-parametric coefficients and integers modulo prime number p $\mathbf{Z}p$ under the condition $p^2 <$ machine word
`"OneParametrGmpZ"`	polynomial arithmetics with one-parametric coefficients over the ring $\mathbf{Z}$ on the basis of the *GMP* library
`"TwoParametrGmpZ"`	polynomial arithmetics with two-parametric coefficients over the ring $\mathbf{Z}$ on the basis of the *GMP* library

`systemType`: specifies type of a system (see 3)

`parametr`: list of parameters for parametric coefficients

`modularShort`: prime number p satisfying $p^2 <$ machine word for the modular coefficients

The class `CoeffInterface` contains the following methods:

`type``()`
returns the string specifying the type of a coefficient.

`isField``()`
returns `True` if the polynomial coefficients form a field, otherwise returns `False`.

`isPseudo``()`
returns `True` if the polynomial coefficients form a ring, otherwise returns `False`.

`parametr``()`
returns the list of parameters

5.2 Coefficient

class Coeff (*coeffInterface, coeff='0'*)
Specifies a coefficient

coeffInterface: defines interface of a coefficient (see 5.1)

coeff: string which initializes a coefficient

The class `Coeff` contains the following methods:

isZero()
returns `True` if a coefficient is zero, otherwise returns `False`.

isOne()
returns `True` if a coefficient is one, otherwise returns `False`.

setZero()
zeroizes a coefficient.

setOne()
makes a coefficient equal to one.

gcd(*coeff*)
returns GCD of a coefficient and coefficient *coeff*.

The coefficients must have the same interface.

diff(*par, deg=1*)
Returns the derivative of a coefficient with respect to the parameter of index *par* raised to the power *deg*.

The class `Coeff` can be an argument of the following functions:

str(*coeff*)
returns representation in the form of coefficient string *coeff* in accordance to the type of system 3.

The Python command **print** works similarly.

+, −, *, /(*coeff1, coeff2*)
These operations return, respectively, the sum, the difference, the product and the quotient of coefficients. To perform the division *coeff2* must divide *coeff1*.

The coefficients must have the same interface.

−(*coeff*)
returns negative coefficient.

+=, −=, *=, /=(*coeff1, coeff2*)
These operations assign the sum, the difference, the product and the quotient of coefficients. For the division *coeff2* must divide *coeff1*.

The coefficients must have the same interface.

A coefficient can be also used in logical expressions. Zero coefficient yields `False` in logical expressions, and other coefficients yield `True`.

A small illustrative program of manipulation with coefficients:

```
import ginv

st = ginv.SystemType("Polynomial")
ic = ginv.CoeffInterface("GmpZ", st)
print ic.type()

print ginv.Coeff(ic, "12221965").gcd(ginv.Coeff(ic, "196196196"))
print ginv.Coeff(ic, "121965") + ginv.Coeff(ic, "196196196")
```

As a result, the following will be printed out:

```
GmpZ
7
196318161
```

6 Polynomials

In `ginv` the following polynomial representations have been implemented:

Polynomial supports bi-directional iterators: `"PolyArray"` (under implementation)

Polynomial does not support bi-directional iterators: `"PolyList"`

6.1 Interface

class `PolyInterface` (*type, systemType, monomInterface, coeffInterface*)
Specifies operations on polynomials

`type`: string defining the polynomial representation

Value of `type`	Polynomial representation
`"PolyList"`	is a singly linked list of the polynomial terms
`"PolyArray"`	is an array of pointers to the polynomial terms

`systemType`: specifies the system type (see 3)

`monomInterface`: specifies the interface of monomials in polynomial (see 4.1)

`coeffInterface`: specifies the interface of coefficients in polynomial (see 5.1)

The class `PolyInterface` contains the following methods:

`type`()
returns the string defining the polynomial representation.

6.2 Polynomial

class `Poly` (*polyInterface, poly='0'*)
Specifies a polynomial

`polyInterface`: specifies the polynomial interface (see 6.1)

`poly`: string initializing a polynomial (see 3)

The class `Poly` contains the following methods:

`length`()
returns the number of terms in a polynomial.

`degree`()
returns the total degree of a polynomial.

This is similar to `degree` of the class `Monom`.

`norm`()
returns the norm of a polynomial.

`lm`()
returns the leading monomial of a polynomial.

`lc`()
returns the leading coefficient of a polynomial.

`setZero`()
zeroizes a polynomial.

pp()
divides a polynomial by its content.

isPp()
returns `True` if the content of a polynomial is 1, otherwise returns `False`.

prolong(*var, deg=1*)
multiplies a polynomial by the variable of index *var* raised to the degree *deg*.

mult(*coeff*)
multiplies a polynomial by a coefficient.

The coefficients of a polynomial and *coeff* must have the same interface.

mult(*monom*)
multiplies a polynomial by a monomial.

Monomials of a polynomial and *monom* must have the same interface.

diff(*par, deg=1*)
returns the derivative of a polynomial with respect to the parameter of index *par* raised to power *deg*.

reduction(*poly*)
reduces a polynomial modulo the polynomial *poly*.

The leading monomial *poly* must divide the leading monomial of a polynomial. The polynomials must have the same interface.

spoly(*poly*)
constructs the *S*-polynomial of a polynomial and the polynomial *poly*.

The polynomials must have the same interface.

The class `Poly` can be an argument of the following functions.

str(*poly*)
returns a string representation of the polynomial string *poly* in accordance to the system type 3.

The Python command **print** works similarly.

cmp(*poly1, poly2*)
returns `1` if `poly1 > poly2`,
returns `0` if `poly1 == poly2`,
returns `-1` if `poly1 < poly2`.

Comparison is done for the leading monomials of polynomials. The polynomials must have the same interface.

<, >, <=, >=, ==, !=(*poly1, poly2*)
In these operations the leading monomials of polynomials are compared in accordance to the order specified. The polynomials must have the same interface.

A polynomial can be used in logical expressions. The zero polynomial yields `False` in logical expressions, and other polynomials yield `True`.

+, /, *(*poly1, poly2*)
These operations return, respectively, the sum, the difference and the product of polynomials.

The polynomials must have the same interface.

+=, /=, *=(*poly1, poly2*)
These operations assign, respectively, the sum, the difference and the product of polynomials to the variable *poly1*.

The polynomials must have the same interface.

len(*poly*)
returns the number of terms in the polynomial *poly*.

`[]` (*poly, i*)
: returns the *i*-th term of a polynomial.

The polynomial is an iterator of 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),
```

As a result, the following will be printed out:

```
(-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 Criteria

In `ginv` the following criteria implemented to detect zero-reducibility of prolongations (S-polynomials):

Without criteria: `"Without"`

Partial involutive criteria: `"C1"`, `"CritPartially"`

Full involutive criteria: `"C1C2C3"`, `"C1C2C3C4"`

Note: Criterion C1 is Buchberger's co-prime criterion. Criteria C2, C3 and C4 in the aggregate are equivalent to the second (chain) Buchberger's criterion. For module orders only the last Buchberger's criterion is applicable.

7.1 Interface

class `WrapInterface` (*type, polyInterface*)
Specifies application of criteria

`type`: string defining the criterion type

Value of `type`	Type of criterion
`"Without"`	Without criteria
`"C1"`	First involutive criterion
`"CritPartially"`	First and second involutive criteria
`"C1C2C3"`	First, second and third involutive criteria
`"C1C2C3C4"`	Full set of criteria

`polyInterface`: Specifies the polynomial interface (see 6.1)

The class `WrapInterface` contains the following methods:

`type``()`
returns the string defining the criterion type.

7.2 Wrapping

class `Wrap` (*wrapInterface, poly*)
Specifies wrapping to organize the criteria application

`wrapInterface`: defines the wrapping interface (see 6.1)

`poly`: polynomial (see 6.2)

The class `Wrap` contains the following methods:

`lm``()`
returns the leading monomial of a polynomial.

`ancestor``()`
returns the ancestor of a given polynomial, i.e. the last if a polynomial with the lowest leading monomial such that the given polynomial has been obtained from that polynomial by a (sequence of) head irreducible non-multiplicative prolongations.

`poly``()`
returns the polynomial.

`multi``()`
returns the number of multiplicative variables.

degProlong()
: returns the degree of variable whose was contributed to obtain the given polynomial as a prolongation of another polynomial.

isProlong()
: returns `True` if a polynomial is prolongation of another polynomial, and `False`, otherwise.

buildProlong()
: returns the list of degrees of variables which were used for prolongation of a given polynomial.

A small program illustrates the work with wrapping of polynomials after the Gröbner basis construction:

```
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])
```

As a result the following will be printed out:

```
          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 Involutive divisions

In `ginv` the following divisions have been implemented:

Admittance of special *Janet* tree structure: `"Janet"`, `"JanetLike"`

Note: The structure of a *Janet* tree allows to find an involutive divisor of a monomial in time $O(d+n)$ where d is the total degree of the monomial monomial and n is the number of variables

8.1 Interface

class `DivisionInterface`(*type, wrapInterface*)
Specifies operations on involutive divisions

`type`: string defining the involutive division type

Value of `type`	Type of involutive division
`"Janet"`	division *Janet*
`"JanetLike"`	division *JanetLike* which, by its structure, is rather similar to *Janet*, but provides generally more compact intermediate and output basis

`wrapInterface`: specifies the criteria applied (see 7.1)

The class `DivisionInterface` contains the following methods:

`type`()
returns the string defining type of an involutive division

8.2 Involutive division

It is planned to provide the user with an option to implement a new involutive division.

9 Algorithms

For constructing Gröbner Bases the following function is used:

basis (*algorithm, divisionInterface, system*)
returns object of class `Basis` which is an involutive basis for system *system*

algorithm: the string defining the algorithm used

Value of **algorithm**	Algorithm used
`"IB"`	algorithm Involutive Basis http://arXiv.org/math.AC/0501111 for any admissible order
`"JBI"`	algorithm Janet Basis I http://arXiv.org/math.AC/0603161 for degree compatible orders (see. 4)
`"JBII/highest"`	algorithm Janet Basis II http://arXiv.org/math.AC/0603161 with the option "highest" in its subalgorithm Update and for degree compatible orders (see. 4)
`"JBII/lowest"`	algorithm Janet Basis II http://arXiv.org/math.AC/0603161 with the option "lowest" in its subalgorithm Update and for degree compatible orders (see. 4)

divisionInterface: specifies an interface for the involutive division (see. 8.1) that is used in the construction of an involutive basis

system: the list representing the polynomials as objects of class `Poly` or their initialization strings (see. 6.2)

9.1 Basis

class Basis
Specifies an involutive basis. The last can be constructed by the function `basis` only.

The class `Basis` contains the following methods:

lengthIB()
returns the number of elements in an involutive basis.

lengthGB()
returns the number of elements in a reduced Gröbner basis.

numCriterion()
returns the number of criteria applied to detect zero-redundancy of prolongations (*S*-polynomials) in the course of the basis construction.

numSpoly()
returns the number of *S*-polynomials computed.

numReduction()
returns the number of of elementary reductions performed.

userTime()
returns the *CPU time* spent for the basis computation.

sysTime()
returns the *system time* spent for the basis computation.

realTime()
returns the *real time* spent for the basis computation.

hilbertPolynomial()
returns the string representing the Hilbert polynomial of the ideal generated by the involutive basis computed.

iterStatistics()
returns the iterator over statistical information on the run of the involutive algorithm. The statistical output is the 4-tuple:

- the time when a new element is inserted in the intermediate basis T
- the number of elements in T
- the number of elements in the set Q containing prolongations to be exemined
- the number of elements in Q with the lowest leading monomial

iterIB()
returns the iterator over the involutive basis represented by the objects in the class `Poly` (see 6.2).

iterGB()
returns the iterator over the Gröbner basis represented by the objects of class `Poly` (see 6.2).

iterLmIB()
returns the iterator over the leading monomials in the involutive basis which are represented by the objects of class `Monom` (see 4.2).

iterLmGB()
returns the iterator over the leading monomials in the Gröbner basis which are represented by the objects of class `Monom` (see 4.2).

find(*monom*)
searches for an involutive divisor for a given monomial among the leading monomials in the basis. *monom* can represent either the string that initializes the monomial or the object in class `Monom` (see 4.2).

The monomials in the basis and *monom* must have the same interface.

nfHead(*poly*)
Provides the head reduction of *poly* modulo the basis. *poly* can represent either the initializing string for the polynomial or the object of class `Poly` (see 6.2).

Polynomials in the basis and *poly* must have the same interface.

isNfHead(*poly*)
Verifies weather *poly* is head reduced modulo the basis and returns `True` if it is the case, and `False` otherwise. *poly* can represent either the initializing string for the polynomial the object of class `Poly` (see 6.2).

Polynomials in the basis and *poly* must have the same interface.

nfTail(*poly*)
Performs the tail reduction of *poly* modulo the basis without the head reduction. *poly* can represent either the initializing string for the polynomial the object of class `Poly` (see 6.2).

Polynomials in the basis and *poly* must have the same interface.

isNfTail(*poly*)
Verifies weather *poly* is tail reduced modulo the basis and returns `True` if it is the case, and `False` otherwise. *poly* can represent either the initializing string for the polynomial the object of class `Poly` (see 6.2).

Polynomials in the basis and *poly* must have the same interface.

The class `Basis` can be an argument of the following functions:

len(*basis*)
returns the number of elements in the involutive basis *basis*.

[](*basis, i*)
returns the *i*-th term in the involutive basis, and the term is represented by objects of class `Wrap` (see 7.2)

The basis is given by the following iterator of 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(),
```

As a result the list of leading monomials in the involutive basis '`x y z^27`' is to be printed out.

A References

B Examples

Examples (benchmarks) are collected in the folder ‘examples’.

B.1 Polynomials

To run a benchmark it is sufficient to open it in **idle** ‘examples.py’ or to enter in the folder of examples the command line

```
python example.py
```

The example choice can be done from the following list:

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

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

In the last line one should edit the file name. Its description is outputted by the command

```
print description
```

One can also get information on an example in the internal folder ‘polynomial’. The example are stored in the *xml*-files compressed with the *gzip* archiver.

B.2 Modules

B.3 Differential equations

B.4 Difference equations

C Comparison with other programs

In this section we present experimental results on comparison of GINV with package JB and Magma (see http://arXiv.org/math.AC/0603161 for references and more details).

The below timing were obtained on the following machines:

JB: 2xOpteron-242 (1.6 Ghz) with 4Gb of RAM running under Gentoo Linux 2004.3 with gcc-3.4.2 compiler.

GINV: Turion-3400 (1.8 Ghz) with 2Gb of RAM running under Gentoo Linux 2005.1 with gcc-3.4.4 compiler.

Magma: dual processor Pentium III (1 Ghz) with 2 GB of RAM running under SuSE Linux 8.0 (kernel 2.4.18-64GB-SMP) with gcc-2.95.3 compiler.

All timings in the table are given in seconds, and (*) shows that the example was not computed because of the memory overflow.

Benchmarking

Example	Algorithm I (JB)	Algorithm I (GINV)	Algorithm II high (GINV)	Algorithm II low (GINV)	Magma V2.11-8	Magma 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
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