Construction of Janet Bases
II. Polynomial Bases

Vladimir P. Gerdt[1], Yuri A. Blinkov[2], Denis A. Yanovich[1]

[1] Laboratory of Information Technologies
Joint Institute for Nuclear Research, 141980 Dubna, Russia
gerdt@jinr.ru yan@cv.jinr.ru
[2] Department of Mathematics and Mechanics
Saratov University, 410071 Saratov, Russia
BlinkovUA@info.sgu.ru

Abstract. In this paper we give a detailed description of an algorithm for computation of polynomial Janet bases and present its implementation in C, C++ and Reduce. This algorithm extends to polynomial ideals the algorithm for computation of monomial Janet bases presented in the first part of our paper and improves the specialization to Janet division of the general algorithm for computation of involutive polynomial bases. The computational efficiency of our codes is compared with that of computer algebra system Singular which provides one of the best implementations of the Buchberger algorithm for computation of Gröbner bases.

1 Introduction

Involutive bases of polynomial ideals [1, 2] and linear differential ideals [3] are Gröbner bases [4, 5] of some special form. Though involutive bases are generally redundant as Gröbner ones, their use makes more accessible the structural information of polynomial and differential ideals and the properties of solutions of algebraic and differential systems. Janet bases came from the pioneering study of differential systems in [6] and may be cited as typical representatives of involutive bases [1]. One more typical representative of involutive bases is Pommaret basis which is the main research tool in formal theory of linear differential equations [7, 8]. When a Pommaret basis is finite it is also a minimal Janet basis [9].

In this paper we present and describe in detail a new optimized version of algorithm [10] for computation of minimal Janet bases for polynomial ideals. The new algorithm extends to polynomials the monomial algorithm described in [11]. As well as for the monomial case we use Janet trees as the basic data structures for intermediate polynomial sets.

We have implemented the optimized algorithm in *C, C++* and *Reduce*. The timings for these three codes are compared with those for the latest release of computer algebra system *Singular* [12] (version 2-0-0) running on the same computer platform. As benchmarks we use the collection in [13] and the data base in [14] which are widely used for testing the Gröbner bases software and numerical routines for solving polynomial systems.

2 Polynomial Janet Bases

Throughout this paper we shall follow where it is possible to definitions, notations and conventions used in [11]. To describe polynomial algorithms and the related data types we need, however, some extra definitions and notations [9].

A divisor u of a monomial v will be called a *proper* divisor if $deg(u) < deg(v)$. The leading monomial of the polynomial $f \in \mathbb{R} \equiv \mathbb{K}[x_1, \ldots, x_n]$ with respect to a given admissible monomial ordering $\succ$ will be denoted by $\mathrm{lm}(f)$. Similarly, the leading term of f as a product of the leading coefficient and the leading monomial will be denoted by $\mathrm{lt}(f)$. If $F \subset \mathbb{R}$ is a polynomial set, then by $\mathrm{lm}(F)$ we denote the leading monomial set for F, and by $Id(F)$ we denote the ideal in R generated by F.

Given a finite polynomial set $F \subset \mathbb{R}$ and a polynomial $f \in F$, the Janet separation of variables into multiplicative and nonmultiplicative is done in accordance with Definition 1 of Janet division in [11] with $u = \mathrm{lm}(f)$ and $U = \mathrm{lm}(F)$. We shall denote by $NM_J(f, F)$ the sets of Janet nonmultiplicative variables for f.

A finite polynomial set F is called *Janet autoreduced* if each term in every $f \in F$ has no Janet divisors among $\mathrm{lm}(F) \setminus \mathrm{lm}(f)$. A polynomial $h \in \mathbb{R}$ is said to be in the *Janet normal form modulo* F if every term in h has no $J-$ divisors in $\mathrm{lm}(F)$. In the following, the definition $NF_J(f, F)$ denotes the Janet normal form of polynomial f modulo F.

A Janet autoreduced set F is called a *Janet basis* of $Id(F)$ if

$$(\forall f \in F)\ (\forall x \in NM_J(f, F))\ \ [\ NF_J(f \cdot x, F) = 0\ ]\,. \tag{1}$$

A Janet basis G of ideal $Id(G)$ is called *minimal* if for any other Janet basis F of the ideal the inclusion $lm(G) \subseteq \mathrm{lm}(F)$ holds.

Janet division is a typical representative of involutive divisions [1] and a minimal Janet basis can be computed by means of general involutive algorithm described in [2, 10] when its input involutive division L is Janet division.

In its simplified form without such important optimizations as avoidance of repeated prolongations and the use of criteria to avoid unnecessary multiplicative reductions the general involutive algorithm may be written in the form of below algorithm **InvolutiveBasis**. The correctness of this algorithm and its termination for any constructive noetherian involutive division [1] follow from those for its improved version [2, 10] and Lemma 1 in [11] which ensures correctness of the selection strategy used in line 6.

The main attributes of algorithm **InvolutiveBasis** are:

- Partition of polynomials in the intermediate basis into two disjoint sets T and Q with shifting from T to Q those polynomials whose leading terms are higher (w.r.t. $\prec$) then the leading term of nonvanishing involutive normal form computed modulo set T for the currently examined element in Q.
- Inclusion in Q all nonmultiplicative prolongations of polynomials in T to be examined.
- Selection of a polynomial in Q to be examined such that its leading term has no proper divisors among leading terms of other polynomials in Q and then elimination of this polynomial from Q.

Algorithm: InvolutiveBasis(F, L, $\prec$)

Input: $F \in \mathbb{R}\setminus\{0\}$, a finite polynomial set; L, an involutive division; $\prec$, an admissible monomial ordering
Output: G, a minimal involutive basis of $Id(F)$

choose $f \in F$ with the lowest $lm(f)$ w.r.t. $\prec$
$G := \{f\}$; $Q := F \setminus G$
do
  $h := 0$
  while $Q \neq \emptyset$ and $h = 0$ **do**
    choose $p \in Q$ with $\mathrm{lm}(p)$ without proper divisors in $\{\mathrm{lm}(q) \mid q \in Q \setminus \{p\}\}$
    $Q := Q \setminus \{p\}$
    $h := NF_L(p, G)$
  od
  if $h \neq 0$ **then**
    $G := G \cup \{h\}$
    for all $\{g \in G \mid \mathrm{lm}(g) \succ \mathrm{lm}(h)\}$ **do**
      $Q := Q \cup \{g\}$; $G := G \setminus \{g\}$
    od
    $Q := Q \cup \{NF_J(g \cdot x, G) \mid g \in G,\, x \in NM_L(g, G),\ NF_L(g \cdot x, G) \neq 0\}$
  fi
od while $Q \neq \emptyset$

– Termination of the algorithm when the set Q becomes empty.

Below we present an advanced form of the above algorithm which preserves the listed attributes and is intended to compute Janet bases. Such a specialization allows one to use appropriate data structures for set T in the form of Janet trees and to design efficient algorithms relied on them. These data structures and algorithms generalize to the polynomial case those we describe in [11] for construction of Janet bases for monomial ideals.

3 Data Structures and Algorithms

Given a finite polynomial set $F \subset \mathbb{R}$ and a monomial ordering $\prec$, we shall endow with every polynomial $p \in F$ the triple structure

$$ps = \{p,\, u,\, vars\}$$

such that

$$\begin{aligned} \mathrm{pol}(ps) &= p \in \mathbb{R} \text{ is a polynomial,} \\ \mathrm{anc}(ps) &= u \in \mathbb{M} \text{ is a monomial,} \\ \mathrm{nmp}(ps) &= vars \text{ is a (possible empty) subset of variables.} \end{aligned}$$

In accordance with the involutivity conditions (1) in the course of construction of a Janet basis all the nonmultiplicative prolongations of intermediate polynomials must be examined with respect to their reducibility by multiplicative prolongations of other elements in the intermediate basis [1]. For this reason the involutive algorithms for construction of Janet bases must sequentially examine nonmultiplicative prolongations. Correspondingly, we shall assign to the second and third elements in the above triple the following values:

- anc(ps) is the lowest (w.r.t. $\prec$) monomial $lm(f)$ ($f \in F$) such that polynomial p was obtained from f by means of the nonmultiplicative prolongations. This means that $\mathrm{lm}(f) \mid \mathrm{lm}(p)$, and we shall call f an *ancestor* of p. If p has no proper *ancestor* in F than we shall assign $\mathrm{anc}(ps) = lm(p)$.
- nmp(ps) is the set of nonmultiplicative variables which were used for generation and examination of nonmultiplcative prolongations of p.

Consider now the structure of Janet tree JT [11] adopted for the polynomial case as a set $JT := \cup\{\nu\}$ of internal nodes and leaves which correspond to a finite nonempty polynomial set endowed with the above triple structure. Now we will assign to every node $\nu \in JT$ the set of four elements $\nu = \{u,\, nd,\, nv,\, ps\}$ as follows:

$$
\begin{array}{lll}
\mathrm{mon}(\nu) & = u & \text{is the monomial assigned to } \nu, \\
\mathrm{ndg}(\nu) & = nd & \text{is the pointer to the next node in degree,} \\
\mathrm{nvr}(\nu) & = nv & \text{is the pointer to the next node in variable,} \\
\mathrm{ps}(\nu) & = ps & \text{is the pointer to the triple } ps.
\end{array}
$$

In addition to our notations and conventions for monomial Janet trees [11] we shall also assign the value *nil* to the pointer ps if ν is an internal node to which we shall not associate any triple.

Then the internal nodes and leaves of a polynomial Janet tree JT are to be characterized by the following states:

$$
\begin{array}{rl}
\text{Internal node:} & ((nv \neq nil) \vee ((nd \neq nil) \wedge (\deg(\mathrm{mon}(nd)) = deg(u) + 1)) \wedge \\
& \wedge (ps = nil) \\
\text{Leaf:} & (nv = nil) \wedge (nd = nil) \wedge (ps \neq nil) \wedge (u = \mathrm{lm}(\mathrm{pol}(ps)))
\end{array}
$$

In doing so, the following algorithm **J-divisor** for the input Janet tree JT and monomial w returns either the leaf triple of JT with the polynomial whose leading term is a $J-$divisor of w or *nil* if there are no such polynomials in JT. The structure of the algorithm is identical to that of its monomial analogue [11] except the output for the existing Janet divisor. In this case the polynomial version of the algorithm returns the corresponding triple whereas the monomial version returns the divisor itself.

Similarly, the polynomial versions of algorithms **OneleafTree** and **J-prolong** are immediate adaptations of their monomial analogs [11] to the above described data structures. By these means the algorithm **OneleafTree** now returns a tree whose extra pointer in the leaf points out at the input triple; the algorithm **J-prolong** augments the input set Q with new triples corresponding to nonmultiplicative prolongations of the leaf polynomials with respect to x_i.

Algorithm: J-divisor(JT, w)

Input: JT, a Janet tree; w, a monomial
Output: the leaf triple p in JT such that $\mathrm{lm}(pol(p)) \mid_J w$ or nil, otherwise

$\nu := \mathrm{root}(JT)$
while $\mathrm{mon}(\nu) \mid w$ **do**
  while $\mathrm{ndg}(\nu)$ **and** $\mathrm{mon}(\mathrm{ndg}(\nu)) \mid w$ **do**
    $\nu := \mathrm{ndg}(\nu)$
  od
  if $\mathrm{nvr}(\nu)$ **then**
    $\nu := \mathrm{nvr}(\nu)$
  elif $\mathrm{ndg}(\nu)$ **then**
    return nil
  else
    return $\mathrm{ps}(\nu)$
  fi
od

Algorithm: OneleafTree(ps, i, w)

Input: ps, the triple structure for a polynomial $\mathrm{pol}(ps)$; i, the index of a variable x_i; w, a divisor of $\mathrm{lm}(\mathrm{pol}(ps))$
Output: $tree$, the one-leaf tree which is rooted at w and has monomial $w\prod_{k=i}^{n} x_k^{deg_k(\mathrm{lm}(\mathrm{pol}(ps)))}$ and the pointer to ps in its leaf

$\nu := \{w,\, nil,\, nil,\, nil\}$
$u := \mathrm{lm}(\mathrm{pol}(ps))$
$tree := \{\nu\}$
while $u \neq w$ **and** $i \leq n$ **do**
  while $w \cdot x_i \mid u$ **do**
    $w := w \cdot x_i$
    $\mathrm{ndg}(\nu) := \{w,\, nil,\, nil,\, nil\}$
    $\nu := \mathrm{ndg}(\nu)$
    $tree := tree \cup \{\nu\}$
  od
  if $u \neq w$ **then**
    $\mathrm{nvr}(\nu) := \{w,\, nil,\, nil,\, nil\}$
    $\nu := \mathrm{nvr}(\nu)$
    $tree := tree \cup \{\nu\}$
    $i := i + 1$
  fi
od
$\mathrm{ps}(\nu) := ps$
return $tree$

Algorithm: J-prolong(jt, i, Q)

Input: jt, a Janet subtree; i, the index of variable x_i which is nonmultiplicative for all the leaf polynomials in jt; Q, a set of triples
Output: the set Q, enlarged with nonmultiplicative prolongations in x_i of the leaf polynomials in jt

```
ν := root(jt)
while ndg(ν) do
   if nvr(nu) then
      J-prolong(jst(ν), i, Q)
   fi
   ν := ndg(ν)
od
if nvr(ν) then
   J-prolong(jst(ν), i, Q)
else
   Q := Q ∪ {pol(ps(ν)) · x_i, anc(ps(ν)), ∅}
   nmp(ps(ν)) := nmp(ps(ν)) ∪ {x_i}
fi
return Q
```

As well as in the monomial case, the algorithms **OneleafTree** and **J-prolong** enter as subalgorithms in the algorithm **J-insert** whose job is also to add new subtrees to the input Janet tree due to the insertion of a new polynomial in the leaf set and to adjust properly the set Q. If the insertion leads to converting of a nonmultiplicative variable into multiplicative then this variable must be eliminated from the relevant third element in the triple (c.f. [10]). This elimination is made in lines 14, 21 and 31.

The next algorithm **J-reduce** takes a set Q of triples and a Janet tree JT as an input and returns set Q with the polynomials in its triples whose leading terms reduced modulo the leaf polynomials in JT. This algorithm enters as a subalgorithm in the main algorithm **JanetBasis** described below. By construction in the latter algorithm, set Q contains the triples with nonmultiplicative prolongations to be examined.

The outer loop in algorithm **J-reduce** for every triple $f \in Q$ verifies in line 2 the existence of Janet divisor among the leading monomials of leaf polynomials in JT. If such a divisor exists, then the inequality $\mathrm{lm}(\mathrm{pol}(f)) \neq \mathrm{anc}(f)$ in line 4 is checked for the selected triple f. This inequality means that $\mathrm{pol}(f)$ has an ancestor in the intermediate basis. These elements of this basis are contained in the leaf triples of JT and there may be another part among the first elements of triples in Q.

Algorithm: J-insert(JT, ps, Q**)**

Input: JT, a Janet tree; ps, the structure with a polynomial pol(ps) such that **J-divisor**(JT, lm(pol(ps))) returns *nil*; Q, a set of triples

Output: $JTree$, the Janet tree obtained from JT by inserting ps; the set Q enlarged with the triples complying with nonmultiplicative prolongations of the leaf polynomials in $JTree$ arising from the insertion of ps

```
i := 1
JTree := JT
ν := root(JTree)
while i ≤ n do
   while deg_i (mon(ν)) < deg_i(lm(pol(ps))) and ndg(ν) do
      ν := ndg(ν)
   od
   if deg_i(mon(ν)) < deg_i(lm(pol(ps))) then
      J-prolong(jst(ν), i, Q)
      jtree :=OneleafTree(ps, i, mon(ν) · x_i)
      ndg(ν) := root(jtree)
      JTree := JTree ∪ jtree
      for j from  i+1 to n do
         nmp(ps)) := nmp(ps) \ {x_j}
      od
      i := n + 1
   else
      if ndg(ν) then
         Q := Q ∪ {pol(ps) · x_i , anc(ps), nmp(ps) ∪ {x_i}}
      else
         nmp(ps) := nmp(ps) \ {x_i}
      fi
      if nvr(ν) then
         i := i + 1
         ν := nvr(ν)
      else
         jtree :=OneleafTree(ps, i + 1, mon(ν))
         ndg(ν) := root(jtree)
         JTree := JTree ∪ jtree
         for j from  i+1 to n do
            nmp(ps)) := nmp(ps) \ {x_j}
         od
         i := n + 1
      fi
   fi
od
return JTree, Q
```

For the prolongation pol(f) the following two criteria are verified in line 5:

- **CriterionI**(f, g) is true iff $anc(f) \cdot \mathrm{anc}(g) \mid \mathrm{lm}(pol(f))$.
- **CriterionII**(f, g) is true iff $\deg(\mathrm{lcm}(\mathrm{anc}(f) \cdot \mathrm{anc}(g))) < \deg(\mathrm{lm}(pol(f)))$.

These two criteria are the Buchberger criteria [4, 5] adapted to the involutive procedure. Indeed, it is easy to see that the first criterion subject to the conditions $g \neq nil$ and $\mathrm{lm}(\mathrm{pol}(f)) \neq \mathrm{anc}(f)$ is true iff anc(f) and anc(g) are coprime. Then, the related $S-$polynomial which arises in the right hand side of assignment in line 11 may be discarded. The second criterion is an involutive analogue of the Buchberger chain criterion [1, 2, 10]. One should note that the the criterion comes into action more rarely than in the course of Buchberger algorithm: many unnecessary $S-$polynomials do not manifest themselves in terms of involutive reduction of nonmultiplicative prolongations.

Algorithm: J-reduce(Q, JT)

Input: Q, a set of triples; JT, a Janet tree
Output: the set Q with leading terms of its polynomials reduced modulo set of the leaf polynomials in JT

1: **for all** $f \in Q$ **do**
2: $\quad g :=$ *J-divisor*$(JT,\ \mathrm{lm}(\mathrm{pol}(f)))$
3: $\quad$ **if** $g \neq nil$ **then**
4: $\quad\quad$ **if** $\mathrm{lm}(\mathrm{pol}(f)) \neq \mathrm{anc}(f)$ **then**
5: $\quad\quad\quad$ **if** *CriterionI*(f, g) **or** *CriterionII*(f, g) **then**
6: $\quad\quad\quad\quad \mathrm{pol}(f) := 0$
7: $\quad\quad\quad\quad g := nil$
8: $\quad\quad\quad$ **fi**
9: $\quad\quad$ **fi**
10: $\quad\quad h := \mathrm{pol}(f)$
11: $\quad\quad$ **while** $g \neq nil$ **do**
12: $\quad\quad\quad h := h - \mathrm{pol}(g) \cdot \mathrm{lt}(h) / \mathrm{lt}(pol(g))$
13: $\quad\quad\quad$ **if** $h = 0$ **then**
14: $\quad\quad\quad\quad g := nil$
15: $\quad\quad\quad$ **else**
16: $\quad\quad\quad\quad g :=$ *J-divisor*$(JT,\ \mathrm{lm}(h))$
17: $\quad\quad\quad$ **fi**
18: $\quad\quad$ **od**
19: $\quad\quad Q := Q \setminus \{f\}$
20: $\quad\quad$ **if** $h \neq 0$ **then**
21: $\quad\quad\quad Q := Q \cup \{h,\ \mathrm{lm}(h),\ \emptyset\}$
22: $\quad\quad$ **fi**
23: $\quad$ **fi**
24: **od**
25: **return** Q

The **while**-loop 11–18 performs the leading term reduction of $h = \mathrm{pol}(f)$, and every reduced and nonvanishing h replaces the triple f in Q with $\{h, lm(h), \emptyset\}$ as is done in lines 19 and 21.

Correctness and termination of algorithm **J-reduce** follow immediately from those for algorithm **J-divisor** [11] and for the leading term reduction in line 12.

The following main algorithm **JanetBasis** is an advanced form of both algorithms **InvolutiveBasis** (Section 2) and that in [10] when their input division is Janet division.

Algorithm: JanetBasis(F, $\prec$)

Input: $F \in \mathbb{R} \setminus \{0\}$, a finite polynomial set; $\prec$, an admissible monomial ordering
Output: G, a minimal Janet basis of $Id(F)$
choose $f \in F$ with the lowest $lm(g)$ w.r.t. $\succ$
$p := \{f, lm(f), \emptyset\}$
$T := \{p\}$
$Q := \{\{q, lm(q), \emptyset\} \mid q \in F \setminus \{f\}\}$
$JT := OneleafTree(p, 1, 1)$
$Q :=$ *J-reduce*$(Q,\ JT)$
while $Q \neq \emptyset$ **do**
    choose $p \in Q$ with $\mathrm{lm}(\mathrm{pol}(p))$ without proper divisors in
            $\{\mathrm{lm}(\mathrm{pol}(q)) \mid q \in Q \setminus \{p\}\}$
    if $\mathrm{lm}(\mathrm{pol}(p)) = 1$ **then**
        return $\{1\}$
    else
        $Q := Q \setminus \{p\}$
        $\mathrm{pol}(p) := NF_J(\mathrm{pol}(p), \{pol(f) \mid f \in T\})$
        if $\mathrm{lm}(\mathrm{pol}(p)) = \mathrm{anc}(p)$ **then**
            for all $\{f \in T \mid \mathrm{lm}(\mathrm{pol}(f)) \succ \mathrm{lm}(\mathrm{pol}(p))\}$ **do**
                $Q := Q \cup \{f\}$
                $T := T \setminus \{f\}$
            od
            $JT := \{\{1, nil, nil, nil\}\}$
            for all $f \in T$ **do**
                J-insert$(JT,\ f,\ Q)$
            od
        fi
    fi
    J-insert$(JT,\ p,\ Q)$
    $T := T \cup p$
    $Q :=$ *J-reduce*$(Q,\ JT)$
od
$G := \{pol(f) \mid f \in T\}$
return G

Initially, in lines 1–6, a Janet tree is constructed with the only leaf polynomial f which has the lowest leading term among the input polynomial set. All other polynomials are placed in Q and their leading monomials are reduced modulo f.

After the initialization step the main loop 7–28 runs while set Q is nonempty. Along with algorithms **InvolutiveBasis** and **MonomimalJanetBasis** [11] the selection strategy of triple $p \in Q$ is used in line 8 providing the lack of proper divisors of $\mathrm{lm}(\mathrm{pol}(p))$ among the leading terms of other polynomials in Q. If the selected polynomial is element in $\mathbb{K} \setminus \{0\}$ that is verified in line 9, then the algorithm returns $\{1\}$ as a Janet basis. Otherwise, p is eliminated from Q in line 12 and the Janet normal form of p modulo the leaf polynomials in JT is computed. Since $\mathrm{lm}(\mathrm{pol}(p))$ is Janet irreducible due to the job of algorithm **J-reduce** in lines 6 and 27, the normal form computation is in fact the tail Janet reduction.

Unlike monomial case when the input set V, which is a monomial analogue [11] of Q and may only be enlarged, in the polynomial case set Q is susceptible to more radical change. This change is made in line 13 of algorithm **InvolutiveBasis** and in line 16 of the current algorithm. Such a transfer of elements $f \in T$ to Q under the condition that $\mathrm{pol}(p)$ is not prolongation of another polynomial in $T \cup Q$, that is verified in line 14, and $\mathrm{lm}(\mathrm{pol}(f)) \succ \mathrm{lm}(\mathrm{pol}(p)$ (line 15) provides minimality of the output Janet basis [2, 10]. After the transfer, the Janet tree JT is fully reconstructed in lines 19–21 for the remaining elements in T if any. Then p is inserted in JT and T in lines 25–26, and polynomials in Q are head reduced in line 27.

For the proof of correctness and termination of algorithm **JanetBasis** we refer to the reasoning exploited in papers [2, 10] and to the above established correctness and termination of algorithm **J-reduce** and algorithms **OneleafTree** and **J-insert** [11].

It should be noted that the leading term reduction of polynomials in Q before their treatment in the **while**-loop decreases the number of transfers of elements in T to Q and related reconstructions of Janet trees. This is a notable optimization of the version of algorithm presented in [10]. Without the pre-reduction of leading terms in Q it may happen that every for a whole group of selected elements in Q causes transfer whereas after the pre-reduction only one of them does it.

4 Implementation

We have implemented the algorithm **JanetBasis** in the symbolic mode language of Reduce 3.7 , C and C++ for the degree-reverse-lexicographical monomial ordering. For this reason we have replaced in our codes the condition of line 15 for moving an element $f \in T$ to Q with $\deg(\mathrm{lm}(\mathrm{pol}(f))) > \deg(\mathrm{lm}(\mathrm{pol}(p)))$. As well as in monomial case [11], the monomials are represented by the array of exponents and polynomials by the lists of their terms. The triple sets Q and T are also represented by lists. In the Reduce code for integer arithmetic, the built-in routines are used whereas for the codes written in C and C++ all the arithmetic operations over integers including modular arithmetic are performed by the GMP MP package version 3.1.1 [15]. In our implementations we use the following selection strategy for elements in Q in line 8 of algorithm **JanetBasis**. First, we chose a triple $p \in Q$ with the minimal $\deg(\mathrm{pol}(p))$. If there are several such triples in Q, then we chose that with the minimal number of terms in $\mathrm{pol}(p)$. If there are still several such triples we select the latest one in the list of Q.

The leftmost column in the below two tables contains a collection of examples extracted[1] from the polynomial systems data bases [13] and [14]. These data bases are widely used as benchmarks for experimentation with Gröbner bases software and with numerical routines for finding roots. In the first table we present the timings in seconds for computing Janet bases with our codes and those for computing Gröbner bases with the computer algebra system *Singular* version 2-0-0 [12]. We have chosen *Singular* for comparison with our codes since it is free and provides one of the best implementations of the Buchberger algorithm.

The timings were measured for the degree-reverse-lexicographical monomial ordering compatible with $x_1 \succ x_2 \succ \cdots \succ x_n$ on a Pentium III 700 Mhz with 546 Mb RAM computer running under Red Hat Linux release 6.2. For compiling the C code we used Intel C/C++ compiler version 5.0.1 beta whereas the C++ code was compiled with gcc compiler version egcs-2.91.66. The columns marked with subscript **m** contain timings for computation modulo prime number 31013. The other columns contain timings for computation over integers. The asterisk $(*)$ marks the cases with running out of memory. As this takes place, the memory bound for running Reduce and C++ is 128 Mb and 546 Mb, respectively. For Singular and our C code a 1 Gb swap is also included. The second table contains the following information on the selected benchmarks:

n is the number of variables.
d is the degree of a system, that is, the maximal total degree of monomials which enter the system.
HP is the Hilbert polynomial of the ideal generated by polynomials in a system. Its degree is the dimension of the ideal. In zero-dimensional case its integer value gives the number of roots of the system.
GB is the number of polynomials in a reduced Gröbner basis.
JB is the number of polynomials in a minimal Janet basis.

Comparison of the timings and memory requirements for our codes with those for Singular shows that in the modular computations *Singular* is significantly faster than our packages with two exceptions only. As regards the computation over integers, our C and C++ codes are faster than Singular for most examples in the table. *Singular* also somewhat more often runs out of memory[2]. Such a variation in comparison of our codes and **Singular** for modular computations and for those over integers enables one to speculate that the algorithm **Janet-Basis** has a more smooth behaviour with respect to the intermediate coefficients swell than the Buchberger algorithm.

[1] We include in our tables only those examples from [13, 14] for which Singular needs on our computer more than half a second to compute Gröbner basis over integers.
[2] Our C++ code runs out of memory for redeco13 nothing but the lack of swap.

example	Reduce	$\mathbf{Reduce_m}$	C	$\mathbf{C_m}$	C++	$\mathbf{C{+}{+}_m}$	Singular	$\mathbf{Singular_m}$
assur44	1 187	23	224	1	238	2	2 774	1
butcher8	15	3	4	0	4	0	10	0
chemequs	52	0	13	0	6	0	7	0
chemkin	205	29	160	3	157	5	7	2
cohn3	*	52	1 109	7	2 373	5	2 167	1
cpdm5	19	14	8	1	8	2	8	1
cyclic6	3	3	1	0	1	0	1	0
cyclic7	2 805	310	527	20	608	37	*	12
cyclic8	*	*	*	751	*	1 415	*	367
d1	235	3	27	0	30	0	38	0
des18_3	4	1	1	0	1	0	1	0
des22_24	12	2	4	0	4	0	4	0
dl	1 151	985	551	114	550	157	242	21
discret3	*	46	*	3	*	6	*	2
eco8	4	4	2	0	2	1	1	0
eco9	23	29	19	3	17	5	14	2
eco10	277	269	196	30	200	54	797	24
eco11	*	2 311	2 242	279	2 229	710	*	404
eco12	*	*	30 950	3 459	24 038	4 444	*	4 215
extcyc5	18	9	5	0	6	1	13	1
extcyc6	*	472	4 566	30	*	58	*	23
extcyc7	*	*	*	12 919	*	28 538	*	3 495
f744	62	46	28	4	29	7	8	1
f855	1 168	309	513	36	466	54	2 603	8
fabrice24	3 276	11	729	1	565	1	30 001	0
filter9	658	10	40	4	46	3	127	1
hairer2	*	296	602	57	386	71	3 986	9
hairer3	*	256	198	35	444	91	3 668	177
hcyclic7	2 566	243	499	21	512	30	21 187	11
hcyclic8	*	*	*	1 207	*	1 432	*	396
hf744	63	48	45	19	28	11	22	2
hf855	*	1 551	4 177	1 608	2 237	430	9 607	55
hietarinta1	1 432	9	20	1	23	2	28	1
i1	*	21	1 240	2	1 301	2	277	1
ilias_k_2	*	240	*	19	*	34	174 576	6
ilias_k_3	*	230	*	28	*	32	116 526	7
ilias12	*	2 337	*	203	*	327	*	52
ilias13	*	497	*	48	*	75	253 351	19
jcf26	6 261	16	1 001	1	1 135	2	3 331	0
katsura7	21	16	7	1	9	2	11	1
katsura8	212	154	102	11	100	21	114	9
katsura9	2 389	1 332	1 181	108	1 170	198	3 230	74
katsura10	*	*	18 313	1 036	21 063	1 964	76 630	651
kin1	1 609	2	164	0	159	0	228	0
kotsireas	601	25	118	2	118	3	38	1
noon7	82	85	88	17	114	39	4	3
noon8	2 736	2 331	2 584	560	4 808	1 765	39	34
noon9	*	*	*	*	*	*	665	445
pinchon1	239	195	197	55	210	91	31	3
rbpl	865	9	1 425	4	1 145	7	5 655	2
rbpl24	3 275	11	728	1	559	1	29 999	0
redcyc6	5	2	1	0	1	0	1	0
redcyc7	*	589	*	38	*	82	*	8
redcyc8	*	*	*	8 110	*	*	*	253
redeco9	10	16	8	3	7	2	2	1
redeco10	96	128	66	12	65	23	21	5
redeco11	873	1 104	606	114	601	207	207	46
redeco12	*	*	6 155	1 108	6 570	2 009	1 979	351
redeco13	*	*	74 765	13 489	*	22 864	19 843	2 986
reimer5	4	5	3	0	2	1	4	0
reimer6	145	144	91	11	89	23	*	38
reimer7	*	7 541	10 801	818	9 790	1 630	*	*
virasoro	91	66	40	3	39	7	161	3

example	**n**	**d**	**HP**	**GB**	**JB**
assur44	8	3	56	87	94
butcher8	8	4	$\frac{1}{2}s^3 + \frac{5}{2}s^2 + 4s + 14$	54	64
chemequs	5	3	16	20	20
chemkin	10	3	80	90	117
cohn3	4	6	$10s + 163$	92	106
cpdm5	5	3	213	77	83
cyclic6	6	6	156	45	46
cyclic7	7	7	924	209	210
cyclic8	8	8	$144s + 760$	372	384
d1	12	3	48	86	104
des18_3	8	3	46	39	104
des22_24	10	2	42	45	129
dl	8	6	468	323	538
discret3	8	2	128	147	164
eco8	8	3	64	59	86
eco9	9	3	128	106	172
eco10	10	3	256	203	336
eco11	11	3	512	389	662
eco12	12	3	1024	743	1329
extcyc5	6	5	350	102	102
extcyc6	7	6	936	242	243
extcyc7	8	7	6468	1371	1373
f744	12	3	$40s - 30$	87	162
f855	14	4	$52s - 36$	148	264
fabrice24	9	3	40	54	55
filter9	9	5	192	153	232
hairer2	13	4	$31s^2 - 40s + 10$	128	453
hairer3	14	5	0	1	1
hcyclic7	8	7	$196s^2 - 546s - 7056$	443	1182
hcyclic8	9	8	$\frac{176}{3}s^3 - 392s^2 + \frac{5944}{3}s - 23936$	1182	4392
hf744	13	3	$\frac{1}{60}s^5 + \frac{17}{12}s^4 + \frac{35}{4}s^3 + \frac{487}{12}s^2 - \frac{10103}{30}s + 570$	204	1244
hf855	15	4	$\frac{1}{360}s^6 + \frac{47}{120}s^5 + \frac{395}{72}s^4 + \frac{425}{24}s^3 - \frac{5962}{45}s^2 - \frac{151}{10}s + 749$	688	4745
hietarinta1	10	3	$\frac{3}{2}s^2 + \frac{23}{2}s + 1$	51	52
i1	12	3	$33s^2 - 40s + 19$	118	147
ilias_k_2	8	5	144	125	185
ilias_k_3	8	5	168	140	205
ilias12	8	5	468	231	314
ilias13	7	5	468	246	285
jcf26	34	4	40	79	80
katsura7	8	2	128	74	79
katsura8	9	2	256	143	151
katsura9	10	2	512	272	287
katsura10	11	2	1024	537	559
kin1	12	3	48	88	120
kotsireas	7	5	$108s - 257$	70	78
noon7	7	3	2173	495	1157
noon8	8	3	6545	1338	3385
noon9	9	3	19665	3682	10001
pinchon1	29	16	78	108	145
rbpl	7	3	$\frac{10}{3}s^3 + 37s^2 - \frac{427}{3}s + 137$	126	309
rbpl24	9	3	40	54	55
redcyc6	7	11	$156s - 588$	21	46
redcyc7	8	13	$924s - 5257$	78	210
redcyc8	9	16	$92s^2 + 474s - 6023$	193	371
redeco10	10	2	256	257	384
redeco11	11	2	512	513	768
redeco12	12	2	1024	1025	1536
redeco13	13	2	2048	2049	3072
reimer5	5	6	144	38	55
reimer6	6	7	576	95	199
reimer7	7	8	2880	227	775
virasoro	8	2	256	128	136

5 Conclusion

The results obtained in this paper give an optimized algorithm for computing the minimal Janet basis of a polynomial ideal and present the first implementation of this algorithm. Its computational efficiency is demonstrated on various examples and compared with that of the Gröbner basis module in *Singular*. The comparison shows the superiority of *Singular* over our codes for modular computations when arithmetic is suppressed. On the other hand our C/C++ codes offer some advantage over *Singular* in computation over integers.

We suggest, and our computer experiments provide support for this suggestion, that the involutive algorithms reveal a more smooth growth of the intermediate coefficients than the Buchberger algorithm. The latter admits much larger arbitrariness in selection and reduction strategies, and various strategies are used in practice to control the intermediate arithmetic. These strategies are essentially heuristic and often tailored for certain classes of examples. In algorithms **InvolutiveBasis** and **JanetBasis** a possible selection of an element in Q is much more restrictive. Furthermore, the leading term pre-reduction of polynomials in Q, as done in algorithm *JanetBases*, facilitates the choice of a triple in Q whose polynomial reduces other polynomials. Generally, the earlier such a polynomial is selected, the less is the growth of intermediate coefficients. The choice we implemented (Section 4) more commonly works in the right direction.

To increase the speed of modular computation we have to pay special attention to internal monomial representation and the related operations over them as it is done in the *Singular* team [16]. Another improvement which may be especially effective in modular computations [17] is planned to be implemented in the next version of our codes. This improvement is suitable for any constructive involutive division [1] and enables one to decrease the number of transfers from set T to set Q performed in algorithms **InvolutiveBasis** and **JanetBasis**.

Since the Pommaret division also admits the tree structure similar to the Janet tree [11] one could draw on the ideas and methods of the present paper to design and implement a special version of algorithm for construction of Pommaret bases. However, this does not seem worthy. Firstly, any finite Pommaret basis is always a minimal Janet basis [9] and, hence, can be computed by algorithm **JanetBasis**. Secondly, having computed a minimal Janet basis it is easy to check whether it is a Pommaret basis: it suffices to verify the coincidence of Janet and Pommaret multiplicative (or nonmultiplicative) variables for every element in the basis [18]. Thirdly, even in the monomial case the construction of finite Pommaret bases with the use of Pommaret trees and Pommaret division does not reveal any notable superiority in the speed of computation over algorithm **MonomialJanetBasis** [11]. Thus, in the polynomial computations it is not to be expected to obtain any gain from the use of Pommaret division instead of Janet division. As to other involutive divisions, the study started in [19] directed on the way to extension of the results of this paper to those divisions.

This work was partially supported by the RFBR grants 00-15-96691, 01-01-00708 and by grant INTAS 99-1222.

References

1. Gerdt, V. P., Blinkov, Yu. A.: Involutive bases of polynomial ideals. *Math. Comp. Simul.* **45** (1998) 519–542
2. Gerdt, V. P., Blinkov, Yu. A.: Minimal involutive bases. *Math. Comp. Simul.* **45** (1998) 543–560
3. Gerdt, V. P.: Completion of linear differential systems to involution. In: *Computer Algebra in Scientific Computing / CASC'99*, V.G. Ganzha, E.W. Mayr and E.V. Vorozhtsov (Eds.), Springer-Verlag, Berlin (1999) 115–137
4. Buchberger, B.: Gröbner bases: an algorithmic method in polynomial ideal theory. In: *Recent Trends in Multidimensional System Theory*, N.K. Bose (ed.), Reidel, Dordrecht (1985) 184–232
5. Becker, T., Weispfenning, V., Kredel, H.: *Gröbner bases. A computational approach to commutative algebra. Graduate Texts in Mathematics* **141**, Springer-Verlag, New York (1993)
6. Janet, M.: *Leçons sur les Systèmes d'Equations aux Dérivées Partielles*, Cahiers Scientifiques, IV, Gauthier-Villars, Paris (1929)
7. Pommaret, J.-F.: *Partial Differential Equations and Group Theory. New Perspectives for Applications*, Kluwer, Dordrecht (1994)
8. Seiler, W. M.: *Analysis and Application of the Formal Theory of Partial Diferential Equations.* PhD thesis, School of Physics and Materials, Lancaster University (1994)
9. Gerdt, V. P.: On the relation between Pommaret and Janet bases. In: *Computer Algebra in Scientific Computing/ CASC 2000*, V.G. Ganzha, E.W. Mayr, E.V. Vorozhtsov (Eds.), Springer-Verlag, Berlin (2000) 167–181
10. Gerdt, V. P.: Involutive division technique: Some generalizations and optimizations, *Zapiski Nauchnykh Seminarov POMI (St.Petersburg)* **258** (1999) 185–206. To be published in *J. Math. Sci.*
11. Gerdt, V. P., Blinkov, Yu. A., Yanovich, D. A.: Construction of Janet bases. I. Monomial bases. This volume.
12. Greuel, G.-M., Pfister, G., Schöenemann, H.: *Singular: A Computer Algebra System for Polynomial Computation*, Department of Mathematics, University of Keiserslautern (2001) http://www.singular.uni-kl.de/Manual/2-0-0/.
13. Bini, D., Mourrain, B.: *Polynomial Test Suite* (1996) http://www-sop.inria.fr/saga/POL.
14. Verschelde, J.: *The Database with Test Examples.* http://www.math.uic.edu/˜jan/demo.html
15. *The GNU Multiple Precision Arithmetic Library.* Edition 3.1.1, 18 September 2000 http://www.gnu.org/manual/gmp/.
16. Bachmann, O., Schönemann, H.: Monomial representations for Gröbner bases computations. In: *Proceedings of ISSAC'98*, ACM Press (1998) 309–321
17. Gerdt, V. P.: On Algorithmic Optimization in Computation of Involutive Bases. In preparation.
18. Seiler, W. M.: A Combinatorial Approach to Involution and δ-regularity. *Preprint*, Universität Mannheim (2000)
19. Blinkov, Yu. A.: The Method of Separative Monomials for Involutive Divisions. *Programming and Computer Software* **3** (2001) 43–45