DEPARTMENT OF MATHEMATICS
TECHNICAL REPORT

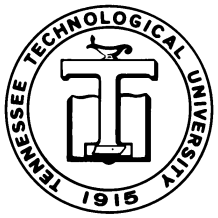
# GRÖBNER BASIS ALGORITHMS FOR GRASSMAN ALGEBRAS IN A MAPLE PACKAGE

MR. TROY BRACHEY

*Tennessee Tech University*

OCTOBER 2008

No. 2008 - 1




TENNESSEE TECHNOLOGICAL UNIVERSITY
Cookeville, TN 38505

# Gröbner Basis Algorithms for Grassmann Algebras in a Maple Package

Troy Brachey

September 30, 2008

**Abstract**

A brief discussion of an approach to calculating Gröbner bases in the Grassmann (exterior) algebra with emphasis on the ideal membership problem is presented. A new package written for computing Gröbner bases in Grassmann algebras for Maple 11 is introduced.

## 1 Introduction

This paper will give details of the algorithms used by the author for the Maple package `TNB`. The `TNB` package is capable of calculating Gröbner bases in the Grassmann algebra. Very little space is devoted to background information and more to the workings of the algorithms themselves.

Two separate types of Gröbner bases will be developed. However, only one will be capable of determining left ideal membership. The steps taken to arrive at each type of Gröbner basis will be similar to the steps taken to compute a commutative Gröbner basis.

## 2 Gröbner Basis Algorithms

This paper is primarily concerned with computations of Gröbner bases within the Grassmann algebra $(\bigwedge V, \wedge)$. The product of any two or more basis elements, $e_i \wedge e_j$ will be denoted as $e_{ij}$. For example, $e_1 \wedge e_2 \wedge e_3 \wedge e_4$ is denoted as $e_{1234}$. The identity element of the algebra is denoted by 1.

**Definition 1.** *[7] The* **super Grassmann or super exterior polynomial algebra** *of order* $(n, m)$ *over a field* $k$*, not of characteristic* 2*, denoted by* $\bigwedge_{n,m}$ *is the* $k$*-linear associative algebra with identity element* 1*, generated by the sets* $A = \{e_1, e_2, \ldots, e_n\}$ *and* $B = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$[1] *subject to the following relations:*

*(i)* $e_i e_j = -e_j e_i$, $1 \leq i, j \leq n$ *(also written as* $e_i \wedge e_j = -e_j \wedge e_i$*)*

[1] In [7], the generators in these sets are referred to, respectively, as "vector" and "scalar" variables.

*(ii)* $\alpha_i\alpha_j = \alpha_j\alpha_i$, $1 \leq i, j \leq m$

*(iii)* $e_i\alpha_j = \alpha_j e_i$, $1 \leq i \leq n, 1 \leq j \leq m$

Note that as a consequence of (i) above $e_i^2 = 0$. Thus there exist nonzero zero divisors. Also observe that $\bigwedge_{n,0}$ is a copy of the Grassmann algebra of order $n$ over the field $k$. Note also that $\bigwedge_{0,m} \cong k[x_1, x_2, \ldots, x_m]$, is the ring of commutative polynomials in $m$ variables over the field $k$. Thus it is not a big leap to see that the commutative techniques should produce similar results on the super Grassmann algebra as they do on $k[x_1, \ldots, x_n]$ since our commutative ring is embedded in the algebra. For the remainder of this paper, let $m = 0$, and so we will be working within the Grassmann algebra.

The quality of being Noetherian is a very important concept in the computation of commutative Gröbner bases as it guarantees termination of the division algorithm. Following the idea that commutative techniques will yield similar results, the quality of being Noetherian needs to be developed for a Grassmann algebra.

**Definition 2.** *A non-commutative ring $R$ is* **left Noetherian** *if it has no infinite ascending chain of left ideals.*

**Lemma 1.** *[7] $\bigwedge_{n,m}$ is left Noetherian. Every left ideal has a finite basis over $\bigwedge_n$.*

Monomial order plays a large role in the computation of noncommutative Gröbner bases just as it does in the commutative case. Choosing a different monomial order can greatly increase the speed of the computation by reducing the number of polynomials needed to generated before a basis is found. Furthermore, in the noncommutative case the monomial order must also be admissible.

**Definition 3.** *A monomial order in $\bigwedge_n$ is said to be* **admissible**, *or* **compatible**, *if:*

*(i)* $m \geq 1$ *for every monomial $m$ in the Grassmann basis;*

*(ii) If $m_2 > m_1$ then $m_l m_2 m_r > m_l m_1 m_r$ for every $m_1, m_2, m_l$, and $m_r$ such that $m_l m_2 m_r \neq 0$ and $m_l m_1 m_r \neq 0$.*

**Example 1.** Consider $\bigwedge_3$ whose basis is given by $[1, e_1, e_2, e_3, e_{12}, e_{13}, e_{23}, e_{123}]$.

- Lex order gives $[e_{123}, e_{12}, e_{13}, e_1, e_{23}, e_2, e_3, 1]$
- InvLex order gives $[e_{123}, e_{23}, e_{13}, e_3, e_{12}, e_2, e_1, 1]$
- RevLex order gives $[1, e_1, e_2, e_{12}, e_3, e_{13}, e_{23}, e_{123}]$
- InvRevLex order gives $[1, e_3, e_2, e_{23}, e_1, e_{13}, e_{12}, e_{123}]$
- Deg[Lex] order gives $[e_{123}, e_{12}, e_{13}, e_{23}, e_1, e_2, e_3, 1]$

- Deg[InvLex] order gives $[e_{123}, e_{23}, e_{13}, e_{12}, e_3, e_2, e_1, 1]$
- InvDeg[Lex] order gives $[1, e_1, e_2, e_3, e_{12}, e_{13}, e_{23}, e_{123}]$

Note that RevLex, InvRevLex, and InvDeg[Lex] orders are not admissible. The admissibility of the monomial order is important when reducing a polynomial and calculating the S-polynomial of two polynomials as termination of the division algorithm only occurs with admissible orders. The S-polynomial in the noncommutative case performs the same function as it does in the commutative case and its definition is analogous to its commutative counterpart.

**Definition 4.** *[4] Let $f, g \in A^r \backslash \{0\}$ with $\mathrm{LM}(f) = x^\alpha e_i$ and $\mathrm{LM}(g) = x^\beta e_j$, respectively. Set $\gamma = (\max(\alpha_1\beta_1), \ldots, \max(\alpha_n\beta_n))$ and define the* **left S-polynomial** *of $f$ and $g$ to be*

$$\mathrm{LeftSpoly}(f,g) = \begin{cases} x^{\gamma-\alpha} f - \dfrac{\mathrm{LC}(x^{\gamma-\alpha} f)}{\mathrm{LC}(x^{\gamma-\beta} g)} x^{\gamma-\beta} g, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

*The left S-polynomial can be simplified if $\mathrm{LM}(g)|\mathrm{LM}(f)$ and $\mathrm{LM}(g) = x^\beta e_i$, $\mathrm{LM}(f) = x^\alpha e_i$, this simplified form is defined as*

$$\mathrm{Spoly}(f,g) = f - \frac{\mathrm{LC}(f)}{\mathrm{LC}(x^{\alpha-\beta} g)} x^{\alpha-\beta} g.$$

The simplified form of the left S-polynomial, Spoly is used for the normal form algorithm but the LeftSpoly form is used for the Gröbner basis algorithm. The use of the Spoly in the normal form algorithm increases efficiency of computation. The definition of the normal form used in this paper and the `TNB` package is as follows.

**Definition 5.** *[4] Let $\mathcal{G}$ denote the set of all finite and ordered subsets $G \subset \bigwedge_n$. then a map $\mathrm{NF} : \bigwedge_n \times \mathcal{G} \to \bigwedge_n$, given by $(f, G) \mapsto \mathrm{NF}(f|G)$, is called a* **(left) normal form** *on $\bigwedge_n$ if, for all $f \in \bigwedge_n, G \in \mathcal{G}$,*

*(a)* $\mathrm{LeftNF}(f|G) \neq 0 \Rightarrow \mathrm{LM}(\mathrm{LeftNF}(f|G)) \notin \langle \mathrm{LT}(G) \rangle$,

*(b)* $f - \mathrm{LeftNF}(f|G) \in \langle G \rangle$.

Note that in line 2 of Algorithm 1 the only polynomials that are being used are ones in which the $\mathrm{LM}(g)|\mathrm{LM}(h)$ and so the simplified form of the S-polynomial can be used. Thus $h$ is reduced by repeatedly computing the Spoly of itself and one of the $g \in G_h$ to eliminate the $\mathrm{LM}(h)$ until all the monomials of $h$ are eliminated or there is no $g \in G$ such that $\mathrm{LM}(g)|\mathrm{LM}(h)$.

**Definition 6.** *[7] $F$ is a* **Gröbner left basis (GLB)** *in $\bigwedge_n$ if for all $g, h_1$ and $h_2$, if $h_1$ and $h_2$ are normal forms of $g$ modulo $F$, then $h_1 = h_2$.*

**Algorithm 1** [4] Left Normal Form LeftNF($f|G$)

1: $h = f$
2: **while** $h \neq 0$ and $G_h = \{g \in G \,|\, \mathrm{LM}(g)|\mathrm{LM}(h)\} \neq \emptyset$ **do**
3: choose $g \in G_h$
4: $h = \mathrm{Spoly}(h, g)$
5: **end while**
6: **return** $h$

**Theorem 1** (Characterization Theorem for GLBs)**.** *[7] The following conditions are equivalent.*

*(i)* $F$ *is a GLB.*

*(ii) If* $f_1, f_2 \in F$, *and* $t \in T_n$ *satisfies* $t \cdot \mathrm{lcm}(\mathrm{LMon}(f_1), \mathrm{LMon}(f_2)) \neq 0$, *then* $\mathrm{LeftNF}(t \cdot \mathrm{LeftSpoly}(f_1, f_2)), F) = 0$.

Based on the characterizations in Theorem 1, Algorithm 2 will compute a GLB for a given list of Grassmann polynomials. Note that lines 6 and 7 of Algorithm 2 ensure that the "if" condition in (ii) of Theorem 1 is met while lines 8 through 17 perform the "then" statement of the Theorem.

**Algorithm 2** [7] Gröbner left basis in $\bigwedge_n$

1: $G := F$, $B = \{\{f_1, f_2\} \,|\, f_1, f_2 \in G; f_1 \neq f_2\}$
2: **while** $B \neq \emptyset$ **do**
3: $\{f_1, f_2\} :=$ an element of $B$
4: $B := B - \{f_1, f_2\}$
5: $g := \mathrm{LeftSpoly}(f_1, f_2)$
6: $V := A - V(\mathrm{lcm}(\mathrm{LMon}(f_1), \mathrm{LMon}(f_2)))$
7: $T := T(V) \cup \{1\}$
8: **while** $T \neq \emptyset$ **do**
9: $t :=$ an element of $T$
10: $T := T - \{t\}$
11: $h := t \cdot g$
12: $h' := N(G, h)$
13: **if** $h' \neq 0$ **then**
14: $B := B \cup \{\{g, h'\} \,|\, g \in G\}$
15: $G := G \cup \{h'\}$
16: **end if**
17: **end while**
18: **end while**
19: **return** $G$

**Definition 7.** *[7]* $F$ *is a* **Gröbner left ideal basis (GLIB)** *if* $f \in LI(F)$ *implies that* $\mathrm{LeftNF}(f, F) = 0$.

**Theorem 2** (Characterization Theorem for GLIBs)**.** *[7] The following statements are equivalent.*

*(i)* $F$ *is a GLIB.*

*(ii) If* $f_1, f_2 \in F$*, then for any* $t$, $\mathrm{LeftNF}(t \cdot f_1, F) = 0$ *and* $\mathrm{LeftNF}(t \cdot \mathrm{LeftSpoly}(f_1, f_2), F) = 0$.

*(iii) For* $f_1, f_2 \in F$ *and for any* $t_1, t_2 \in T(V)$ *satisfying the conditions*

$$t_1 \cdot \mathrm{LTerm}(f_1) = 0 \text{ and } t_2 \cdot \mathrm{lcm}(\mathrm{LTerm}(f_1), \mathrm{LTerm}(f_2)) \neq 0,$$

*then* $\mathrm{LeftNF}(t_1 \cdot f_1, F) = 0$ *and* $\mathrm{LeftNF}(t_2 \cdot \mathrm{LeftSpoly}(f_1, f_2), F) = 0$.

Theorem 2 leads directly to Algorithm 3 for computing GLIBs.

**Algorithm 3** [7] Gröbner left ideal basis in $\bigwedge_n$

```
1:  G := F, H := F, B = {{f1, f2} | f1, f2 ∈ G; f1 ≠ f2}
2:  while H ≠ ∅ do
3:     f := an element of H
4:     H := H − {f}
5:     W := V(LMon(f))
6:     S := {t | t ∈ T(A), V(t) ∩ W ≠ ∅}
7:     while S ≠ ∅ do
8:        u := an element of S
9:        k := u · f
10:       k' := NF(G, k)
11:       if k' ≠ 0 then
12:          H := H ∪ {k'}
13:          G := G ∪ {k'}
14:          B := B ∪ {{g, k'} | g ∈ G}
15:       end if
16:    end while
17: end while
18: G:=GLB(G)
19: return G
```

Algorithm 3 is based on Theorem 2 part (iii). Observe that lines 3 through 6 fulfill the "if" condition on $t_1$ and lines 7 through 16 fulfill the "then" statement concerning $t_1$. The role of $t_2$ is fulfilled by line 18 where Algorithm 2 is called. Thus it is easy to see that every GLIB is a GLB. A GLB will not determine left ideal membership. However, the GLIB will determine the left ideal membership making it far more useful to work with. Thus the GLB's importance comes from its contribution to the GLIB computation.

# 3 The TNB Package

The `TNB` package is a package of procedures written for Maple 11 that can compute Gröbner left bases and Gröbner left ideal bases in a Grassmann algebra. When the package is loaded, 20 functions are exported.

```
> with(TNB);
```

```
TNB - package for computing Groebner bases in Grassmann algebras
TNB version delta 0.3 (March 12, 2008)
20 functions exported
```

[*Deg*, *InvDeg*, *InvLex*, *InvRevLex*, *LCoeff*, *LMon*, *LTerm*, *Lex*, *NF*, *RevLex*, *TLSpolyG*, *TglbG*, *TglibG*, *TlcmG*, *TmingbG*, *idealpoly*, *isadmissible*, *makelist*, *pairs*, *testing*]

Several procedures are from the package `GfG` [2]. These procedures are the monomial orderings and procedures that use the monomial orderings to find the leading term, monomial, and coefficients. The procedures are *Deg*, *InvDeg*, *InvLex*, *InvRevLex*, *LCoeff*, *LMon*, *LTerm*, *Lex*, *RevLex*, *isadmissible*. They are included with permission [2].

The remaining procedures were written specifically for the `TNB` package. These procedures are described below.

1. Procedure *TlcmG* returns the lcm of two Grassmann monomials, $A$ and $B$. An optional third argument can be used to return a list $[a, \mathrm{lcm}, b]$ where $a$ and $b$ are signed factors of the lcm, that is $a \wedge A = \mathrm{lcm}(A, B)$ and $b \wedge B = \mathrm{lcm}(A, B)$.

2. Procedure *NF* reduces a single Grassmann polynomial $p_1$ by a list of Grassmann polynomials $P$ following Algorithm 1.

3. Procedure *TLSpolyG* takes two polynomials $f$ and $g$ and following Definition 4 returns LeftSpoly$(f, g)$.

4. Procedure *TglbG* will compute a Gröbner left basis according to Theorem 1 using Algorithm 2.

5. Procedure *TglibG* computes a Gröbner left ideal basis according to Theorem 2 based on Algorithm 3.

6. Procedure *TmingbG* will compute a minimal Gröbner basis. An optional third argument can be used to have the original list returned on the first line, the rejected polynomials on the second line, and the minimal Gröbner basis on the third line of the output. Assigning a name to the procedure will result in the minimal Gröbner basis being assigned that name.

7. Procedure *pairs* generates all possible pairs of elements from a given list.

8. Procedure *makelist* makes a random list of $N$ Clifford polynomials of type clibasmon, climon, or clipolynom in a Clifford or Grassmann algebra over

a space of dimension $n$. The procedure does not return a zero polynomial. If one of its randomly generated polynomials happens to be zero, it recursively calls itself until it returns exactly $N$ nonzero such polynomials. This procedure is useful for testing purposes.

9. Procedure *idealpoly* creates a random polynomial (Clifford or Grassmann, depending on the second argument) in an one-sided ideal $J$ - left or right, depending on the third argument- and generated by polynomials supplied as the first list $F$. The second argument is expected to be wedge - for the wedge product in a Grassmann algebra, or cmul - for the Clifford product in a Clifford algebra. In either case, the algebra is considered over a space of dimension equal to the largest index found in the list $F$. An output of this procedure contains three items, in the following order:

   (a) A list of random coefficients.

   (b) A list of generators (it is a permuted list $F$ due to removal of duplicates in the first line of the procedure when Maple may and usually does permute elements of $F$).

   (c) Random polynomial. Assigning a name to this procedure will result in the random polynomial being assigned to that name.

   This procedure is useful for testing purposes.

10. Procedure *testing* takes a list of polynomials $F$ and for each $t \in T(V)$ and computes the product $t \wedge f_i$ for $i = 1, \ldots, n$ where $n$ is the number of polynomials in the given list $F$ then finds the LeftNF$(t \wedge f_i|F)$. This is based on Theorem 2. It is useful in determining whether a given Gröbner basis is a GLIB.

Package `TNB` also relies upon the `CLIFFORD` package [1]. The `CLIFFORD` package provides the basis monomials and exterior, or wedge, product which is used for multiplication of Grassmann monomials. Type checking of arguments passed to `TNB` procedures is also provided by the `CLIFFORD` package. When loading, the `TNB` package automatically loads the `CLIFFORD` package and the `BIGEBRA` package which is part of the `CLIFFORD` library.

# 4 Applications and Examples

This section will show examples of the `TNB` package in use. In many examples the results are compared to those from `Plural` which computes Gröbner bases for Grassmann algebras using the methods described in [4]. `Plural` is more flexible than `TNB` package in that it can perform Gröbner basis calculations for many other algebras than only the Grassmann algebra. However, `TNB` holds a pedagogical value as it allows us to understand differences between GLB and GLIB bases. It is important to note that the `TNB` package will return monic polynomials where `Plural` does not [6].

**Example 2.** [7] Given $f_1 = e_{56} - e_{13}$ and $f_2 = e_{45} - e_{23}$ procedures TglibG and TglibG will be used to compute a Gröbner left basis and a Gröbner left ideal basis, respectively.

```
> restart:with(linalg):B:=diag(0$9):
  eval(Clifford:-makealiases(9,'ordered')):
  with(SINGULARPLURALlink):
  with(TNB):
> f1:=e56-e13;f2:=e45-e23;F:=[f1,f2];
```

$$f1 := e56 - e13$$

$$f2 := e45 - e23$$

$$F := [e56 - e13,\ e45 - e23]$$

```
> TglbG(F,Deg[InvLex]);

Computed 10 S-polynomials among 5 polynomials in Groebner basis and
needed to compute 10
```

$$[e1236,\ e1234,\ e236 - e134,\ e56 - e13,\ e45 - e23]$$

```
> GLIB:=TglibG(F,Deg[InvLex]);

Computed 21 S-polynomials among 7 polynomials in Groebner basis and
needed to compute 21

 This procedure took 6.630 seconds to run.
```

$$GLIB := [e236 - e134,\ e136,\ e235,\ e135,\ e234,\ e56 - e13,\ e45 - e23]$$

The results just obtained agree with results from `Plural` .

```
> Pgb:=PLURALforGlink(F,6,ls,[e1,e2,e3,e4,e5,e6],
                      input_for_Singular,input_for_Maple,'infty','d');
```

$$Pgb := [e56 - e13,\ e45 - e23,\ -e134 + e236,\ e235,\ e234,\ e136,\ e135]$$

```
> evalb(convert(GLIB,set)=convert(Pgb,set));
```

$$true$$

In the last line the two lists of generators from the Gröbner bases `GLIB` and `Pgb` are converted into sets to be easily compared and the result is *true*, that is they are identical.

**Example 3.** Consider the following list of polynomials. The `w` is the syntax used by `CLIFFORD` to represent the wedge product of the Grassmann algebra. Similarly `CLIFFORD` uses the *Id* symbol to represent the identity element of the algebra.

```
> p1:=Id+2*e1+3*e3+e1we3;
  p2:=e1+3*e2we3;
  p3:=e2+e1;
  p4:=e1we2we3;
  P:=[p1,p2,p3,p4];
```

$$p1 := Id + 2\,e1 + 3\,e3 + e13$$

$$p2 := e1 + 3\,e23$$

$$p3 := e2 + e1$$

$$p4 := e123$$

$$P := [Id + 2\,e1 + 3\,e3 + e13,\ e1 + 3\,e23,\ e2 + e1,\ e123]$$

First a GLIB will be computed by `TNB`. This Gröbner basis is capable of determining ideal membership.

```
> Tgb:=TglibG(P,Deg[Lex]);
```

```
Computed 21 S-polynomials among 7 polynomials in Groebner basis and
needed to compute 21

 This procedure took 3.495 seconds to run.
```

$$Tgb := [e123,\ Id + 2\,e1 + 3\,e3 + e13,\ \tfrac{1}{3}e1 + e23,\ e2 + e1,\ e2,\ e3,\ Id]$$

Now a minimal Gröbner basis will be computed from the GLIB `Tgb`. The minimal Gröbner basis computed here is analogous to the minimal Gröbner basis from the commutative case [3]. Since its input is the GLIB `Tgb` the minimal Gröbner basis will also be able to determine membership.

```
> Tmgb:=TmingbG(Tgb,Deg[Lex]);
```

$$Tmgb := [Id]$$

The minimal Gröbner basis `Tmgb` has only one generator, the $Id$ element of the algebra. This indicates that the ideal generated by $P$ is the whole algebra. Next `Plural` will compute a reduced Gröbner basis for the polynomials $p_1, p_2, p_3$, and $p_4$ contained in the list $P$.

```
> Pgb:=PLURALforGlink(P,0,dp,[e1,e2,e3],input_for_Singular,
  input_for_Maple,'infty','d');
```

$$Pgb := [Id]$$

```
> Tmgb=Pgb;
```

$$[Id] = [Id]$$

Finally, the results obtained show that in this case the minimal Gröbner basis computed by `TNB` is the same as the reduced Gröbner basis `Plural` computed. It is important to note that the minimal Gröbner basis is not always the same as the reduced Gröbner basis in either the commutative or non-commutative cases.

**Example 4.** By altering the polynomials from Example 3 more interesting results can be obtained. For this example, the $Id$ monomial will be removed from $p_1$. As in the previous example, a GLIB will be computed first and then a minimal Gröbner basis will be computed from it.

```
> p1:=e1+e3+e13;
  p2:=e1+e23;
  p3:=e2+e1;
  p4:=e123;
  P:=[p1,p2,p3,p4];
```

$$p1 := e1 + e3 + e13$$

$$p2 := e1 + e23$$

$$p3 := e2 + e1$$

$$p4 := e123$$

$$P := [e1 + e3 + e13,\ e1 + e23,\ e2 + e1,\ e123]$$

```
> GBt:=TglibG(P,Deg[Lex]);
```

```
Computed 15 S-polynomials among 6 polynomials in Groebner basis and
needed to compute 15

 This procedure took 3.494 seconds to run.
```

$$GBt := [e123,\ e1 + e3 + e13,\ e1 + e23,\ e2 + e1,\ -e3 + e2,\ e3]$$

```
> GBtm:=TmingbG(GBt,Deg[Lex]);
```

$$GBtm := [e2 + e1,\ -e3 + e2,\ e3]$$

The minimal Gröbner basis $GBtm$ is capable of determining ideal membership. Now `Plural` will compute a reduced Gröbner basis.

```
> Pgb:=PLURALforGlink(P,0,dp,[e1,e2,e3],input_for_Singular,
  input_for_Maple,'infty','d');
```

$$Pgb := [e3,\ e2,\ e1]$$

```
> GBtm<>Pgb;
```

$$[e2 + e1,\ -e3 + e2,\ e3] \neq [e3,\ e2,\ e1]$$

Note that the lists `GBtm` and `Pgb` are not identical. The difference is due to the fact that Gröbner basis returned by `TNB` is a minimal Gröbner basis and the one returned by `Plural` is a reduced Gröbner basis, which is a concept that is also applicable in the non-commutative case [5]. To show that they are in fact equal, it is sufficient to show that each of the generators of one ideal is contained in the other. This can be accomplished by using the `NF` procedure to reduce each generator with respect to the generators of the other ideal. Since `Pgb` is a reduced Gröbner basis it can determine ideal membership.

```
> for i from 1 to 3 do NF(GBtm[i],Pgb,Deg[Lex]) end do;
```

$$0,\ 0,\ 0$$

This shows that every generator in the list `GBtm` is in the ideal generated by the polynomials (in this case monomials) of `Pgb`.

```
> for i from 1 to 3 do NF(Pgb[i],GBtm,Deg[Lex]) end do;
```

$$0,\ 0,\ 0$$

It is important to recall that a minimal Gröbner basis given by the procedure `TmingbG` is capable of determining ideal membership since it is a reduced form of the GLIB given by the procedure `TglibG`. This last result shows that every generator in the list `Pgb` belongs to the ideal generated by the polynomials in the list `GBtm`. Each generator reduces to zero modulo the generators of the other ideal. Thus, as ideals, `GBtm` $\subseteq$ `Pgb` and `Pgb` $\subseteq$ `GBtm` and therefore, `GBtm=Pgb` as desired. This means that the left ideals generated by the two lists are equal.

**Example 5.** For this example the polynomials will be randomly generated. Once again a GLIB will be computed first, then a minimal Gröbner basis and the results compared to the reduced Gröbner basis produced by the `Plural` system. The `TNB` package is capable of calculating Gröbner bases in Grassmann algebras of up to nine variables.

```
> N:=4:numofpols:=4:
  i:='i':
  P:=[]:
  for i from 1 to numofpols do
  p||i:=0:
  end do:
  for i from 1 to numofpols do
  while p||i=0 or p||i=Id do
  p||i:=rd_clipolynom(N):
  end do:
  P:=[op(P),p||i];
  end do;
  vars:=[e||(seq(i,i=1..N))];
```

$$P := [7\,Id - 8\,e14]$$

$$P := [7\,Id - 8\,e14,\ 6\,Id + 6\,e13]$$

$$P := [7\,Id - 8\,e14,\ 6\,Id + 6\,e13,\ -4\,e1 + 5\,e3]$$

$$P := [7Id - 8e14,\ 6Id + 6e13,\ -4e1 + 5e3,\ Id - 2e34 - 2e23 + e14 + e13 + e3]$$

$$vars := [e1,\ e2,\ e3,\ e4]$$

```
> GB1:=TglibG(P,Deg[Lex]);

Computed 15 S-polynomials among 6 polynomials in Groebner basis and
needed to compute 15

 This procedure took 17.784 seconds to run.
```

$$GB1 := [Id - 2\,e34 - 2\,e23 + e14 + e13 + e3,$$
$$Id + e13,\ -\tfrac{7}{8}\,Id + e14,\ e1 - \tfrac{5}{4}\,e3,\ e3,\ Id]$$

```
> GB1m:=TmingbG(TNBglib,Deg[Lex]);
```

$$GB1m := [Id]$$

```
> Pgb:=PLURALforGlink(P,0,dp,vars,input_for_Singular,
  input_for_Maple,'infty','d');
```

$$Pgb := [Id]$$

```
> evalb(GB1m=Pgb);
```

$$true$$

The last result shows that the two lists are equivalent. Thus the ideal generated by the four polynomials is the whole algebra. Even though the polynomials were restricted to degree at most 4, any degree polynomial will be in the left ideal generated by them. This is from the fact that the reduced Gröbner basis is only the *Id* element and so any polynomial in any degree algebra will be a multiple of the generator.

**Example 6.** This example will use randomly generated polynomials of a higher degree. Instead of using four polynomials of degree up to 4, there will be two polynomials of unto degree 6 generated. That is the polynomials will be from the Grassmann algebra $\bigwedge_6$. The same procedure will be used as before.

```
> N:=6:numofpols:=2:
  i:='i':
  P:=[]:
  for i from 1 to numofpols do
  p||i:=0:
  end do:
  for i from 1 to numofpols do
  while p||i=0 or p||i=Id do
  p||i:=rd_clipolynom(N):
  end do:
  P:=[op(P),p||i];
  end do;
  vars:=[e||(seq(i,i=1..N))];
```

$$P := [-e13 + 3\,e46 - e6 - e2356]$$

$$P := [-e13 + 3\,e46 - e6 - e2356,\ e2]$$

$$vars := [e1,\ e2,\ e3,\ e4,\ e5,\ e6]$$

```
> GB1:=TglibG(P,Deg[Lex]);

Computed 45 S-polynomials among 10 polynomials in Groebner basis and
needed to compute 45

 This procedure took 16.769 seconds to run.
```

$$GB1 := [e1456 + \tfrac{1}{3}e156,\ e13 - 3\,e46 + e6 + e2356,\ e135 + 3\,e456 + e56,$$
$$e146 - \tfrac{1}{3}e16, e156,\ e346 - \tfrac{1}{3}e36,$$
$$e13 - 3\,e46 + e6,\ e16,\ e36,\ e2]$$

```
> GB1m:=TmingbG(GB1,Deg[Lex]);
```

$$GB1m := [e13 - 3\,e46 + e6,\ e16,\ e36,\ e2]$$

```
> Pgb:=PLURALforGlink(P,0,dp,vars,input_for_Singular,
input_for_Maple,'infty','d');
```

$$Pgb := [e2,\ e36,\ e16,\ e13 - 3\,e46 + e6]$$

```
> evalb(convert(GB1m,set)=convert(Pgb,set));
```

$$true$$

In the last result the two lists `GB1m` and `Pgb` were converted to sets and compared since Maple usually puts lists in random order making it difficult to compare longer lists. The result is *true* and so the two sets are equal and thus the lists are equal. This means that the ideal generated by the polynomials that were randomly generated can be generated by the four polynomials given in `GB1m` and `Pgb`. Thus `TNB` gave the same result as `Plural` did.

**Example 7.** This example will demonstrate that the GLB does not solve the membership problem but the GLIB will. A few extra lines of code also allow one to follow the computation of the GLB by listing out the S-polynomials as they are computed and the new polynomials as they are added to the GLB $G$.

```
> f1:=e2456-e3;f2:=e14-e36;f3:=e1256-e13;
  F:=[f1,f2,f3];
```

$$f1 := e2456 - e3$$

$$f2 := e14 - e36$$

$$f3 := e1256 - e13$$

$$F := [e2456 - e3,\ e14 - e36,\ e1256 - e13]$$

```
> GLB:=TglbG(F,Deg[InvLex]);

LSpoly(e2456-e3,e1256-e13)=-e13+e134
Add polynomial -e13+e134 to GLB
LSpoly(e2456-e3,-e14+e36)=0
LSpoly(e1256-e13,-e14+e36)=0
LSpoly(e2456-e3,-e13+e134)=e12356
LSpoly(e1256-e13,-e13+e134)=e12356
LSpoly(-e14+e36,-e13+e134)=-e136

Computed 6 S-polynomials among 4 polynomials in Groebner basis and
needed to compute 6
```

$$GLB := [e2456 - e3,\ e1256 - e13,\ -e13 + e134,\ -e14 + e36]$$

```
> GLIB:=TglibG(F,Deg[InvLex]);

Computed 28 S-polynomials among 8 polynomials in Groebner basis and
needed to compute 28

 This procedure took 6.703 seconds to run.
```

$$GLIB := [e2456 - e3,\ e1256 - e13,\ -e14 + e36,\ e35,\ e34,\ e14,\ e23,\ e13]$$

```
> f:=idealpoly(F,w,left);
```

$$[2e1234 + 2e156 + 2e456 - e46,\ -6Id - 6e2356,$$
$$-2Id - 2e4 - 2e2 - 2e1 - 2e6]$$

$$[e14 - e36,\ e2456 - e3,\ e1256 - e13]$$

$$f := -6\,e2456 + 6\,e3 - 2\,e1256 - 2\,e12456 + 2\,e13 + 2\,e134 - 2\,e123 + 2\,e136$$

```
> NF(f,GLB,Deg[InvLex]);
```

$$-2\,e123$$

```
> NF(f,GLIB,Deg[InvLex]);
```

$$0$$

Note that $f$ does not reduce to 0 modulo `GLB` even though $f1, f2$, and $f3$ all reduce to 0 modulo `GLB`. However, $f$ does reduce to 0 modulo `GLIB`. This shows that the GLIB solves the ideal membership problem while the GLB does not.

# 5 Conclusion

The above algorithms and examples show how the package `TNB` computes GLB and GLIB bases in the Grassmann algebra. In particular, we have explicitly

demonstrated that the GLB bases do not solve the ideal membership problem whereas the GLIB bases do. Once a Gröbner basis is computed using the `TglibG` procedure a polynomial can be tested for membership in the ideal generated by that Gröbner basis using the `NF` procedure. If the `NF` procedure returns 0, then the polynomial belongs to the ideal, otherwise it does not.

# References


[1] R. Abłamowicz and B. Fauser, 'CLIFFORD' - Maple 11 package for Clifford algebra computations, ver. 11, `http://math.tntech.edu/rafal/cliff11/`, 2008.

[2] R. Abłamowicz and B. Fauser, 'GfG' - Maple 11 package for Gröbner Bases for Grassmann Algebras, ver. 0.5, `http://math.tntech.edu/rafal/GfG/`, 2008.

[3] D. Cox, J. Little, D. O'Shea, *Ideals, Varieties, and Algorithms,* $2^{nd}$ *ed.* Springer, NY, 2000.

[4] V. Levandovskyy and H. Schönemann, Plural – A Computer Algebra System for Noncommutative Polynomial Algebras, *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, New York, 2003, 176–183.

[5] T. Mora, An introduction to commutative and non-commutative Gröbner Bases, *J. Theoretical Comp. Sci.* **135** (1994), 131–174.

[6] Singular with Plural, `http://www.singular.uni-kl.de/`, 2008.

[7] T. Stokes, Gröbner Bases in Exterior Algebra, *J. Automated Reasoning* **6** (1990), 233–250.


## Acknowledgment


Many thanks to Dr. Rafał Abłamowicz for his assistance in the research and preparation of this material.