FGB: C interface

Jean-Charles Faugere
Jean-Charles.Faugere@inria.fr

This documentation shows how to use the C interface of FGb to compute a Gröbner basis. In section 1of this document, a global overview of the calling sequence is given. Basically there are fours 4 steps:

1. Define the polynomial ring $\mathbb{K}[x_1,\ldots,x_n]$ and the admissible monomial ordering.
2. Convert/Creat the input polynomials (see section 2)
3. Run the computation with various optional parameters (see section 4)
4. Print/convert the output polynomials (see section 3)

1 Global Overview

1.1 Specification

The goal of the library is to compute a Gröbner basis of a list of polynomials$[f_1,\ldots,f_m]$ in $\mathbb{K}[x_1,\ldots,x_n]$ where

$$\begin{array}{ll} \mathbb{K}=\mathbb{Q} & \text{if } p=0 \\ \mathbb{K}=\mathbb{F}_p & \text{if } p\neq 0 \text{ and } p<2^{16} \end{array}$$

Hence, the ground field is defined by one integer parameter: p.
The admissible monomial ordering is a block ordering:

$$\mathrm{DRL}(x_1,\ldots,x_{k_1}) \gg \mathrm{DRL}(x_{k_1+1},\ldots,x_n)$$

Hence, the monomial ordering is defined by two integer parameters: k_1 and $k_2=n-k_1$.
Remark that the usual grevlex/DRL monomial ordering can be obtained by setting $k_1=n$ and $k_2=0$

1.2 Installation and example of a valid source code

The FGb interface is available at: `http://www-polsys.lip6.fr/~jcf/FGb/index.html`

Install the corresponding .tar.gz archive.
We refer to the file `README.txt` for compilation instructions.
To understand this manual, the best thing to do is to compile the source code files provided: several Gröbner bases computations over a prime field or over the integers. Edit the files `main.c`, `gb1.c` (computation over $\mathbb{Q}$) and `gb2.c` (computation in $\mathbb{F}_{65521}$) . For instance (Mac OS X):

```
make
gcc -m64  -I ../../protocol -I ../../int  -c main.c
gcc -m64  -I ../../protocol -I ../../int  -c gb1.c
gcc -m64  -I ../../protocol -I ../../int  -c gb2.c
g++ -o fgbdemo main.o gb1.o gb2.o -Lmacosx -lfgb -lfgbexp -lgb -lgbexp -lminpoly
                                  -lminpolyvgf -lgmp -lm -fopenmp
```

Then the file `fgbdemo` can executed:

```
./fgbdemo 1
```

1.3 Requirements

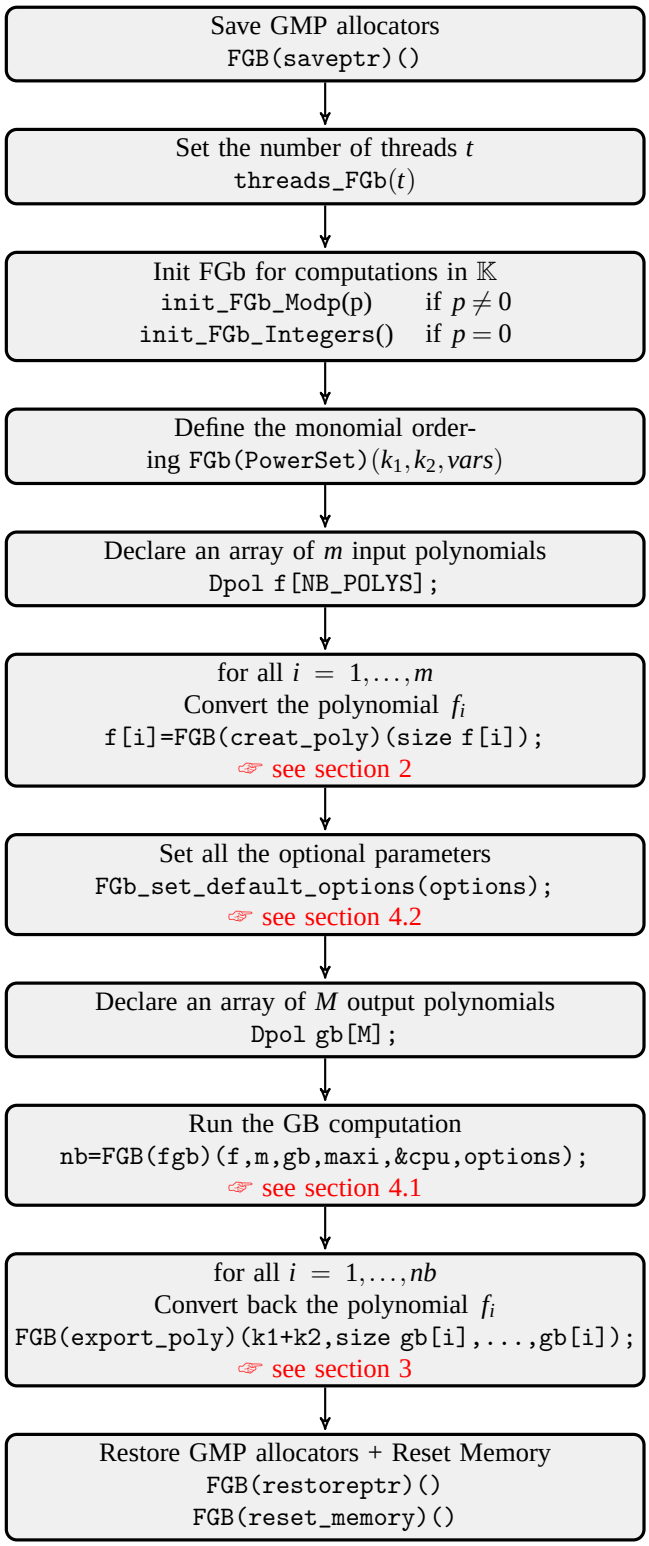
- C/C++ compiler (gcc/g++)
- openmp should be installed (used for multi-thread computations)
- Since gmp integers are used to input/output the coefficients of the polynomials it is mandatory to have the GMP library available during the compilation. For instance, somewhere in the code there is an inclusion of the gmp.h header file:

  ```
  #include "gmp.h"
  ```

  Also, the GMP library should also be available when linking:

  ```
  g++  -o main main.o gb1.o gb2.o -Lmacosx [...] -lgmp -lm -fopenmp
  ```

1.4 C interface: global view



1.5 Structure of the code

In the following code `I32` (resp. `UI32`) is C data type equivalent to a signed (resp. unsigned) integer 32 bits word; it is defined in the header file `call_fgb.h`. Also `Dpol` or `Dpol_INT` are virtual data type used to store the input/output polynomials. The user has no direct access to the internal representation of these polynomials.

1.5.1 Define the polynomial ring (modulo a prime p)

```
/* We use the library to compute a GB modulo a prime number: we set LIBMODE to 1 */
#define LIBMODE 1

/* We use the library as a package so we do not need to define functions */
#define CALL_FGB_DO_NOT_DEFINE

#include "call_fgb.h"

/* Maximum number of polynomials in the basis */
#define FGb_MAXI_BASE 100000

void gb()
{
  Dpol input_basis[FGb_MAXI_BASE];
  Dpol output_basis[FGb_MAXI_BASE];
  I32 m=0;
  const int nb_vars=5;
  char* vars[5]={"x1","x2","x3","x4","x5"}; /* name of the variables (can be anything) */

  FGB(saveptr)(); /* First thing to do : GMP orignal memory allocators are saved */
  threads_FGb(1);
  init_FGb_Modp(65521); /* init FGb for modular computations */
  /* We compute in GF(65521)[x1,x2,x3,x4,x5] */

  FGB(PowerSet)(4,1,vars);
  /* ================================================ */
  /* Create the first polynomial and store the result in the input_basis array */
  input_basis[m++]=FGB(creat_poly)(4); /* number of monomials in the polynomial (here 4) */
  ....
```

1.5.2 Define the polynomial ring (over the rationals)

```
/* The following macro should be 2 to call FGb over the integers */
#define LIBMODE 2

/* To remove verbosity define the following macro to 1 and define your own I/O function (see at the
#define USE_MY_OWN_IO 0


/* ------------------------------------------------------------------- */

#if USE_MY_OWN_IO
#define CALL_FGB_DO_NOT_DEFINE
#endif /*  USE_MY_OWN_IO */
```

```
#include "call_fgb.h"

/* Only needed if GMP is used to import the coefficients */
#include "gmp.h"

/* Maximum number of polynomials in the basis */
#define FGb_MAXI_BASE 100000

void gb()
{
  Dpol input_basis[FGb_MAXI_BASE];
  Dpol output_basis[FGb_MAXI_BASE];
  I32 m=0;
  const int nb_vars=6;
  char* vars[6]={"x1","x2","x3","x4","x5","x6"}; /* name of the variables (can be anything) */

  FGB(saveptr)(); /* First thing to do : GMP origmal memory allocators are saved */

  init_FGb_Integers(); /* init FGb for integers computation  */

  FGB(PowerSet)(nb_vars,0,vars); /* Define the monomial ordering: DRL(k1,k2) where
                                    k1 is the size of the 1st block of variables
                                    k2 is the size of the 2nd block of variables
                                    and vars is the name of the variable
                                 */
  threads_FGb(1);
  /* Creat the first polynomial  */
  prev=FGB(creat_poly)(2); /* number of monomials in the polynomial (here 2 so that poly = monom0 + m
  input_basis[global_nb++]=prev;  /* fill the array of input polynomials with the first polynomial *
  ....
```

1.5.3 Run the computation

We assume that the initialization of the Polynomial Ring has already been done previously. The input polynomials $f_1, \ldots, f_m$ have been stored in a array of polynomials.

```
    int nb;
    const int n_input=5; /* we have 5 input polynomials */
    SFGB_Options Opt;
    FGB_Options options=&Opt;

    /* set default options (by default we compute a GB) */
    FGb_set_default_options(options);

    /* overide some default parameters */
    options->_env._force_elim=0; /* if force_elim=1 then the computation will return only
                                    the result of the elimination (is is mandatory to
                                    define a monomial ordering DRL(k1,k2) with k2>0 ) */
    options->_env._index=500000; /* This is is the maximal size of the matrices generated by F4
                                    you can increase this value according to your memory */
```

```
options->_verb=1; /* display useful infos during the computation */

/* Other parameters :
   t0 is the CPU time (reference to a double)
*/
nb=FGB(fgb)(input_basis,n_input,output_basis,FGb_MAXI_BASE,&t0,options);
```

1.5.4 Extract and display the polynomials in the final GB

```
 /* Import the internal representation of the i-th polynomial computed by FGb */
for(i=0;i<nb;i++)
 {
   const I32 nb_mons=FGB(nb_terms)(output_basis[i]); /* Number of Monomials */
   UI32* Mons=(UI32*)(malloc(sizeof(UI32)*nb_vars*nb_mons));
   I32* Cfs=(I32*)(malloc(sizeof(I32)*nb_mons));
   FGB(export_poly)(nb_vars,nb_mons,Mons,Cfs,output_basis[i]);
   I32 j;
   for(j=0;j<nb_mons;j++)
     {

       I32 k,is_one=1;
       UI32* ei=Mons+j*nb_vars;

       if (j>0) printf("+");
       printf("%d",Cfs[j]);

       for(k=0;k<nb_vars;k++)
         if (ei[k])
           printf("*%s^%u",vars[k],ei[k]);
     }
 }
```

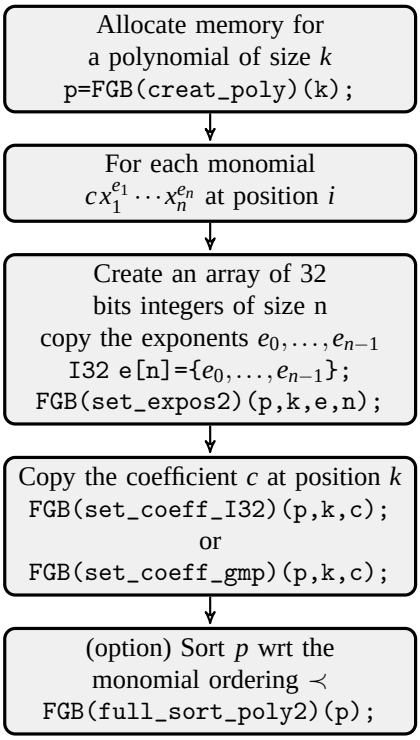
2 Converting polynomials (from C to FGb)

Each multivariate polynomial f is a sum of monomials:

$$f = \sum_{i=0}^{k-1} c_i \, x_0^{e_{i,0}} \cdots x_{n-1}^{e_{i,n-1}}$$

Remark: when the field is $\mathbb{K} = \mathbb{Q}$, the coefficients c_i are to be taken in $\mathbb{Z}$. In this case, for simplicity, we assume that c_i is a GMP integer.

2.1 Global view over $\mathbb{F}_p/\mathbb{Q}$



2.2 Example over $\mathbb{F}_p$

We assume that the polynomial ring $\mathbb{F}_{65521}[x_1,x_2,x_3,x_4,x_5]$ has already been initialized. Suppose that m-th polynomial (m is integer variable already defined in the code) is

$$p = 2x_2x_3 + 2x_2x_5 + 2x_1x_4 - x_4 \text{ in } \mathbb{F}_{65521}[x_1,x_2,x_3,x_4,x_5]$$

Hence p is a sum of 4 monomials and we will create in FGb this polynomial monomial by monomial. For the purpose of this we will proceed from the left to the right but the order is not important since we will sort the polynomial according the monomial ordering at the end of the process. The m-th polynomial will be created:

```
m=...; /* integer variable */
p=FGB(creat_poly)(4); /* number of monomials in the polynomial (here 4) */
input_basis[m++]=p;  /* fill the array of input polynomials with the corresponding polynomial */
/* Create the first monomial  = coef * term */
{
  /* the first term: power product x1^e[1]*...*xn^e[n] */
  I32 e[5]={0,1,1,0,0};  /* monomial: x2*x3 */
  FGB(set_expos2)(p,0,e,nb_vars); /* arguments:
                                     0: the first monomial
                                     nb_vars: the number of variables
                                   */
}
/* the first coefficient (here 2) */
FGB(set_coeff_I32)(p,0,2); /* arguments:
                                     0: the first coefficient
                                     2: value of the coefficient (modulo the prime number)
```

```
                                    */

  /* the second term: power product x1^e[1]*...*xn^e[n] */
  {
    I32 e[5]={0,1,0,0,1}; /* x2*x5 */
    FGB(set_expos2)(p,1,e,nb_vars);  /* second monomial */
  }
  FGB(set_coeff_I32)(p,1,2); /* second coefficient */
  {
    I32 e[5]={1,0,0,1,0}; /* x1*x4 */
    FGB(set_expos2)(p,2,e,nb_vars); /* third monomial */
  }
  FGB(set_coeff_I32)(p,2,2); /* third coefficient */
  {
    I32 e[5]={0,0,0,1,0}; /* x4 */
    FGB(set_expos2)(p,3,e,nb_vars); /* fourth monomial */
  }
  FGB(set_coeff_I32)(p,3,65520); /* fourth coefficient */

  FGB(full_sort_poly2)(p); /* it is recommended to sort each polynomial */
```

2.3 Example over $\mathbb{Q}$

$$p = -x_2x_4x_6 + 17x_3x_4x_5 \text{ in } \mathbb{Q}[x_1,x_2,x_3,x_4,x_5,x_6]$$

We assume that the polynomial ring $\mathbb{Q}[x_1,x_2,x_3,x_4,x_5,x_6]$ has already been initialized.

```
     p=FGB(creat_poly)(2); /* number of monomials in the polynomial (here 2 so that poly
                              = monom0 + monom1 */
  input_basis[global_nb++]=p;  /* fill the array of input polynomials with the first polynomial */

  /* the first monomial  = coef * terme */
  {
    /* the first term: power product x1^e[1]*...*xn^e[n] */
    I32 e[6]={0,1,0,1,0,1}; /* x2*x4*x6 */
    FGB(set_expos2)(p,0,e,nb_vars); /* first monomial monom0 */
  }
  /* the first coefficient (here -1) */
  {
    mpz_t u;
    mpz_init_set_str(u,"-1", 10);
    FGB(set_coeff_gmp)(p,0,u);
   }
  /* the second term: power product x1^e[1]*...*xn^e[n] */
  {
    I32 e[6]={0,0,1,1,1,0}; /* x3*x4*x5 */
    FGB(set_expos2)(p,1,e,nb_vars); /* second monomial monom1 */
  }
  /* Coefficient of the second monomial */
  {
    mpz_t u;
    mpz_init_set_str(u,"17", 10);
```

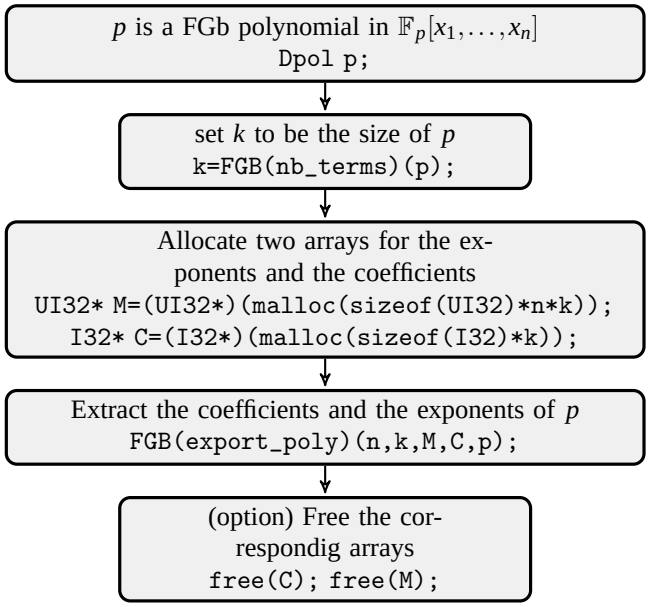
```
    FGB(set_coeff_gmp)(p,1,u);
  }
  FGB(full_sort_poly2)(p);/* it is recommended to sort each polynomial */
```

3 Converting polynomials (from FGb to C)

Each multivariate polynomial f is a sum of monomials:

$$f = \sum_{i=0}^{k-1} c_i \, x_0^{e_{i,0}} \cdots x_{n-1}^{e_{i,n-1}}$$

3.1 Global view over $\mathbb{F}_p$

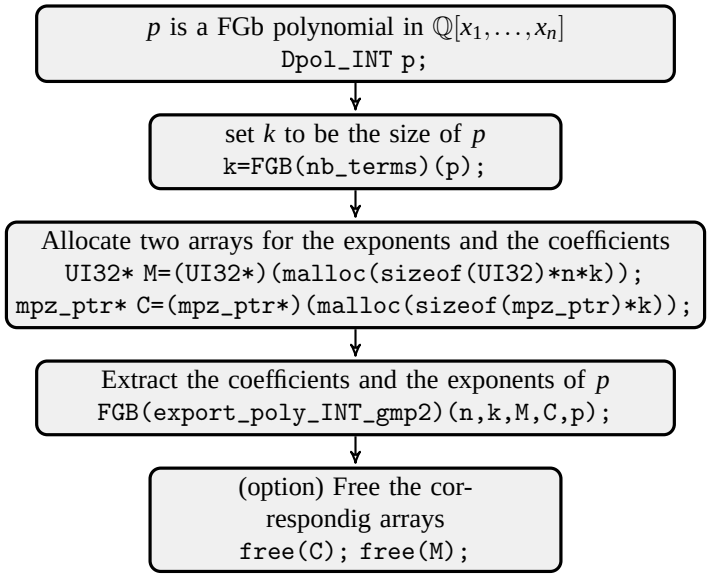


As an example, the following code will import and display the i-th polynomial computed by FGb (`i` is an integer variable and `output_basis` is an array of polynomials):

```
            {
              const I32 nb_mons=FGB(nb_terms)(output_basis[i]); /* Number of Monomials */
/* we allocate an array for the coefficients and the monomials */
              UI32* Mons=(UI32*)(malloc(sizeof(UI32)*nb_vars*nb_mons));
              I32* Cfs=(I32*)(malloc(sizeof(I32)*nb_mons));
              FGB(export_poly)(nb_vars,nb_mons,Mons,Cfs,output_basis[i]);
              I32 j;
              for(j=0;j<nb_mons;j++)
                {

                  I32 k,is_one=1;
                  UI32* ei=Mons+j*nb_vars;

                  if (j>0) printf("+");
                  printf("%d",Cfs[j]);
```

```
                for(k=0;k<nb_vars;k++)
                  if (ei[k])
                    {
                      if (ei[k] == 1)
                        printf("*%s",vars[k]);
                      else
                        printf("*%s^%u",vars[k],ei[k]);
                      is_one=0;
                    }
                if (is_one)
                  printf("*1");
              }
/* free the two arrays */

            free(Cfs);
            free(Mons);
```

3.2 Global view over $\mathbb{Q}$



Warning !
Do not modify the elements inside the array C (not a copy).

In a similar way we also display the i-th polynomial computed by FGb:

```
/* Import the internal representation of each polynomial computed by FGb */
{
  const I32 nb_mons=FGB(nb_terms)(output_basis[i]); /* Number of Monomials */
  /* temporary allocation of two arrays for the monomials and the coefficients */
  UI32* Mons=(UI32*)(malloc(sizeof(UI32)*nb_vars*nb_mons));
  mpz_ptr* cfs=(mpz_ptr*)(malloc(nb_mons*sizeof(mpz_ptr)));
  FGB(export_poly_INT_gmp2)(nb_vars,nb_mons,cfs,Mons,output_basis[i]);

  I32 j;
  for(j=0;j<nb_mons;j++)
```

```
{
  I32 k,is_one=1;
  UI32* ei=Mons+j*nb_vars;

  if (j>0) printf("+");
  /* use GMP function to print the coefficient */
  mpz_out_str(stdout,10,cfs[j]);

  for(k=0;k<nb_vars;k++)
    if (ei[k])
      {
        if (ei[k] == 1)
          printf("*%s",vars[k]);
        else
          printf("*%s^%u",vars[k],ei[k]);
        is_one=0;
      }
  if (is_one)
    printf("*1");
}
 /* the two arrays are no longer useful */
free(Mons);
free(cfs);
```

4 Running the Gröbner basis computation

4.1 Main call

The main call to FGb is a very simple function:

UI32 `FGB(fgb)`(Dpol* input,UI32 n_input,Dpol* output,UI32 n_output,double* cpu,FGB_Options options)

The value returned by this function is the number of polynomials in the final Gröbner basis (an unsigned integer). The input parameters are:

- `input` an array of polynomials
- `n_input` the number of input polynomials
- `output` an array of polynomials
- `n_output` the maximal size of the final Gröbner basis
- a reference to a double (used to store the CPU time)
- the last parameter `options` is a structure with all the optional parameters (see next section).

4.2 Options

`SFGB_Options` is a structure defined to store all the parameters.

```
SFGB_Options options;
```

The default parameters are sufficient in most cases. Hence it is always a good practice to reset the parameters to their default values:

```
FGb_set_default_options(&options);
```

We list the most important parameters:

- `options._env._index` is used to specify the maximum of the matrices during the Gröbner basis computations (F_4 algorithm). For instance to increase this default value to 10^6 we define:

  ```
  options._env._index=1000000;
  ```

- when an elimination ordering is chosen at the beginning of the computation

  $$\mathrm{DRL}(x_1,\ldots,x_{k_1}) \gg \mathrm{DRL}(x_{k_1+1},\ldots,x_n)$$

  and `options._env._force_elim` is to 1 then only the eliminated Gröbner basis is returned:

  $$GB \cap \mathbb{K}[x_{k_1+1},\ldots,x_n]$$

- `options._verb=1` is used to display some useful infos during the computation:

  ```
  [2](10x21)100%/[3](34x46)100%/[4](56x64)100%/[5](53x60)100%/[6](54x60)100%/[7](58x64)100%/
  [8](59x66)100%/[9](57x66)100%/[10](49x61)100%/[11](59x70)100%/[12](60x71)100%/[13](64x74)
  100%/[14](65x75)100%/[15](66x76)100.0%/100%/[16](65x77)100%/[17](56x68)100%/
  Elapsed time: 0.00s/0.00s (Total/Symb+Linalg)
  ```

  - `[d]` is the current degree in F_4
  - `(n x m)` is the size of the matrix generated by F_4
  - `30%40%50%` is useful to track progress of a lengthy computation (linear algebra step).

- The `FGB(fgb)` procedure is the unique procedure to run a computation. The specific nature of the computation (Grobner bases, Normal Forms, Sparse-FGLM, Hilbert function, ...) has to be specified by setting `options._env._compute`; possible values are to be chosen in the following list:

```
typedef enum {
  FGB_COMPUTE_GBASIS=1,
  FGB_COMPUTE_RRFORM,
  FGB_COMPUTE_RRFORM_SQFR,
  FGB_COMPUTE_MINPOLY,
  FGB_COMPUTE_MINPOLY_SQFR,
  FGB_COMPUTE_GBASIS_NF,
  FGB_COMPUTE_GBASIS_NF_RECOMPUTE,
  FGB_COMPUTE_RADICAL_STRATEG1,
  FGB_COMPUTE_RADICAL_STRATEG2,
  FGB_COMPUTE_NOP,
  FGB_COMPUTE_GBASIS_MULTI,
  FGB_COMPUTE_MATRIXN,
  FGB_COMPUTE_MATRIXN_COMBI
} FGB_TYP_COMPUTE;
```

When the defaults options are used

```
FGb_set_default_options(options);
```

this automatically set the value of `options._env._compute` to `FGB_COMPUTE_GBASIS` and a Gröbner basis of the input polynomials is computed.
For instance to compute the NormalForm of two polynomials f and g wrt an already computed Gröbner basis G: we have to put in array of polynomials `Dpol F[N]`, the two input polynomials and the Gröbner basis:

$$\begin{aligned} F[0] &:= f \\ F[1] &:= g \\ F[2] &:= G[0] \\ F[3] &:= G[1] \\ &\dots \end{aligned}$$

In this case we have set the following optional parameters:

```
options._env._compute=FGB_COMPUTE_GBASIS_NF;
options._env._nb=2; /* If nenv->_nb=2 it means that we want to compute a normalForm of

                       [f,g] wrt to G=[f2,f3,...]
                       Please note that G should be a Groebner Basis */
```