The PyM Poly functions	
Function signature	**Description**
hohner(A,x)	Given a list A and an element x, it returns a_0*x**r+a_1*x**(r-1)+....+a_{r-1}*x +a_r
polynomial(A,x)	Given a list A and an element x, it returns a_0+a_1*x+...+a_{r-1}*x**{r-1}+a_r*x**r

degree(f,v)
degree(f)
	Returns the degree of the polynomial f in the principal variable. If v is given, it returns the degree of the polynomial f in the variable v.

trailing_degree(f)
trailing_degree(f,v)
	Returns the degree of the trailing term of f in the principal variable. If v is given, it returns the degree of the trailing tem of f in the variable v. The trailing term is defined as the non null term of a polynomial with the least degree

trailing_coeff(f)
trailing_coeff(f,v)
	Returns the coefficient of the trailing term of f in the principal variable. If v is given, it returns the coefficient of the trailing tem of f in the variable v. The trailing term is defined as the non null term of a polynomial with the least degree.

trailing_term(f)
trailing_term(f,v)
	Returns the trailing term of f in the principal variable. If v is given, it returns the trailing tem of f in the variable v. The trailing term is defined as the non null term of a polynomial with the least degree.

leading_coeff(f)
leading_coeff(f,v)
	Returns the coefficient of the leading term of f in the principal variable. If v is given, it returns the coefficient of the leading tem of f in the variable v. The leading term is defined as the non null term of a polynomial with the greatest degree.

leading_term(f)
leading_term(f,v)
	Returns theleading term of f in the principal variable. If v is given, it returns the leading tem of f in the variable v. The leading term is defined as the non null term of a polynomial with the greatest degree.

constant_coeff(f)
constant_coeff(f,v)
	Returns the coefficient of the constant term of f in the principal variable. If v is given, it returns the coefficient of the constant tem of f in the variable v. The constant term is defined as the monomial of degree zero.

zero_degree_term(f)	Return the constant monomial of f in the principal variable.

derivative(f)
derivative(f,v)
partial_derivative(f,v)
	Returns the derivative of the polynomial f in the principal variable. If variable v is given, it returns the partial derivative of f in the variable v.

coeffs (p)
coeffs(p,var)
coeffs_dec(p)
coeffs_dec(p,var)
coeffs_inc(p)
coeffs_inc(p,var)
	If p is a polynomial, coeffs(p) gives the list of its coefficients, from highest to lowest degree. The multivariate polynomials are considered as univariate polynomials in the last variable with coefficients in the polynomial ring of the previous variables. coeffs(p,var) does the same but with respect to the variable var.
	coeffs_dec does the same as coeffs. coeffs_inc does the same as coeffs but it returns the coefficients from lowest to highest degree.

sugiyama(r0,r1,t)	The function sugiyama (r0, r1, t) is a key subroutine of the BMS decoder and works as follows. Given non-zero univariate polynomials r0 and r1 in the variable z such that 0 < deg(r1) ≤ deg(r0), and an integer t such that t ≤ deg(r1), sugiyama returns a pair [v,r] of univariate polynomials in z such that r ≡ v·r1 mod r0 and deg(r) < t.

PR(K,*symb,name)
polynomial_ring(K,*symb,name)
multivariate_polynomial_ring(K,*symb,name)
	It creates the polynomial ring over K. It can be given an undefined number of symbols as strings and the function returns the corresponding symbols.

bivariate_polynomial_ring(K,x,y,name)	It creates the polynomial ring over K of two variables x and y. It returns the symbols x and y.
multivariate_named_polynomial_ring(K,x,d,name)	It creates the polynomial ring over K with d symbols x1...xd.

get_subpolynomial_degree(f,d)
hpart = get_subpolynomial_degree
	return the parts of degree d of the polynomial f as a polynomial

variables(f)
variable(f)
	variables(f) returns the list of variables of f.
	variable(f) returns the principal variable of f.

The PyM Poly functions	
Function signature	**Description**

mdegree(f)
total_degree(f)
tdegree = mdegree
	returns the degree of f counting all the variables

skeleton(f)	Given a polynomial f, it returns a pair of list. The first one is the coefficients of f and the second one the indeces of the variables.
lexicographic_order()	Represents all the polynomials with lexicographic order in the variables.

expand_polynomial_on()
expand_polynoial_off()
	On: It represents all the polynomials with all the monomials expanded. Off: It represent all the polynomials grouping its terms by degree of the principal variable. By default the option is off.

polynomial_variables_order(L)	Given a list of variables L, assign the order of the variables in the polynomials. If a variable is not given, it goes bellow the given ones.
add_variable(f,x)	Let's suppose f is a polynomial that depends of the variables [x1,..,xm], then the return will be the same polynomial that depends on [x,x1,...,xm]. If f already depends on x, it returns a copy of f

evaluate(f,vars,value)
evaluate(f,values)
	Given an element f, it returns f evaluating the list of variables vars into its corresponding element of the list value. If this function is called without the vars argument, then, it returns a list of polynomials where f principal variable is evaluated at the elements of the list values.
	Examples:
	f = 1+3*x+9*x**3. evaluate(f,0) = 1, evaluate(f,[1])=[13], evaluate(f,[1,2])=[13,79]
	g = 5+3*x+x**2+y , evaluate(g,0) = 5+3*x+x**2, evaluate(g,[0,1]) = [5+3*x+x**2,6+3*x+x**2], evaluate(g,x,1)=9+y, evaluate(g,[x,y],[1,2])= 11, evaluate (g,z,1)=5+3*x+x**2+y, evaluate(g,[x,z],[1,2])= 9+y

symbol(*L)	Given a list L of string names, it returns a list of the corresponding symbols.

coeff_inverse_polynomial(k,p)
list_coeff_inverse_polynomial(n,p)
	Given a polynomial f and an integer n, the list_coeff_inverse_polynomial function return the coefficient of a polynomial g of degree n such that f*g = 1 + o(x**n+1). coeff_inverse_polynomial returns the k-th coefficient of g.

zero_positions(f,a)	Given a polynomial f and a list of elements a, it returns a list with the indeces of the elements in a that f(ai) = 0
homogenize(f,z='')	It homogenize the polynomial f using the variable z.
newton(j)	It returns the polynomial 1/j! x*(x-1)*...*(x-j+1)

krawtchouk(k,n,q)
krawtchouk(k,n)
	It computes the Krawtchouk polynomial of a q-linear code (n,k). By default q = 2.

collect(f,var)	It returns the (multivariate) polynomial f as a polynomial in the variable var.
expand_poly()	It represent all polynomials totally expanded as sums of monomials.
not_expand_poly()	It represent all polynomials as a polynomial in its principal variable.
is_polynomial(f)	It returns if f is of type polynomial, monomial or symbol.

cyclic_reduction(f,n)
cyclic_product(f,g,n)
	Given a polynomial f = a0+a1*x+....+ a_m*x**m, cyclic_reduction returns the polynomial by reducing all the monomials exponents in Zn, i.e, transforming the monomials a_k*x**k to a_k*x**(k%n).
	cyclic_product computes the product of the polynomials f and g and then applies the reduction of order n.

polynomial_parts(p)
monomial_parts(m)
	Given a polynomial, polynomial_parts returns a list of pairs (c,m) in decressing degree order where c is the coefficient and m is the monomial term.
	Given a monomial, monomial_parts returns (ind,var,c) where ind is the list of indexes of the monomial, var is the list of variables and c is the coefficient.

monomial(X,E)	Given a list of variables [v_1,...,v_n] and a list of indexes [i_1,..., i_n], it returns the monomial v_1^i_1*...*v_n^i_n. To give a coefficient, you can add that coefficient in the list of variables and put that index equal 1.

The PyM Poly functions	
Function signature	**Description**

monomial_substitution(m,D)
polynomial_substitution(p,D)
	Given a monomial m and a dictionary D of the form {v,w}, it returns the monomial m substituing the variable v for w.
	Given a polynomial p and a dictionary D of the form {v,w}, it returns the polynomial p substituing the variable v for w.

is_square_free(f)	Given a polynomial f, it returns if the gcd(f,f')=1.

distinct_degree_factoring(f,K)
DDF = distinct_degree_factoring(f,K)
	Given a squarefree monic polynomial f in Fq[x] of degre n>0, it delivers a *distinct-degree* factoring f = f1⋯fr of f (the fj have different degrees). See VzG, example 14.5

equal_degree_splitting(f,r)
EDS = equal_degree_splitting
	If f is monic polynomial of Fq[X], q odd, and f = f1⋯fm, where the fj are distinct irreducible of degree r, it finds a proper monic factor of f (if m>1), or error otherwise.

equal_degree_factoring(f,r)
EDF = equal_degree_factoring
	Given a squarefree monic polynomial f in Fq[x] of degre n>0, q odd, such that all irreducible factor of f have the same degree r, it yields the irreducible factors of f. See VzG, Algorithm 14.10

factor(f,K)
factor(f)
	The function factorize the polynomial f over K. By default K is Zn(2)

find_roots(f,K)
find_roots(f)
roots = find_roots
	It returns the list of roots of a polynomial f over K. By default K is the domain of the coefficients of f.

cyclotomic_polynomial(n,K)
cyclotomic_polynomial(n)
	It returns the cyclotomical polynomial of degree n over the domain K. By default K is Z_.

cyclotomic_polynomial_moebius(n)	n-th cyclotomic polynomial Q_n. It is a different version than the one called cyclotomic_polynomial

cyclotomic_splitting(n,K)
cyclotomic_splitting(n)	

minimal_polynomial(x,K,T)	It returns the minimal polynomial of the element x over the domain K with variable T.

fast_cyclotomic(n,x)
fast_cyclotomic(n)
	A faster version of cyclotomic_polynomial(n). The parameter x gives the name of the variable. By default is 'x'

lagrange_interpolators(J)	If J is a set of elements of a field and r = card(J), this function returns a dictionary {j:Lj for j in J}, where Lj are polynomials of degree r-1 such that L_j(i) =delta_i^j for all i in J and j in 1..r.
lagrange_scalar_interpolators(J)	Works like lagrange_interpolators(J), but the returned dictionary is {j:Lj(0) for j in J}. The code, however, is optimized so that the values Lj(0) are not obtained directly by evaluating the Lj at zero but computed directly with operations in the domani on the elements of J.