Tarea final AC

Algoritmo de la división

Código

```
DIVISIÓN(f_1, ..., f_2, f):
 1  q_1 := 0; ...; q_s := 0; r := 0 p := f
 2  while p ≠ 0 do
 3        i := 1
 4        divisionoccurred=false
 5        while i ≤ s and divisionoccurred=false do
 6              if lt(f_i) divides lt(p) then
 7                    q_i := q_i + lt(p)/ lt(f_i)
 8                    p := p − (lt(p)/ op(f_i))f_i
 9                    divisionoccurred=true
10              else
11                    i := i + 1
12        if divisionoccurred = false then
13              r := r + lt(p) p := p − lt(p)
14  return q_1, ..., q_s, r
```

Ejemplo del libro

Sean $f_1 = xy - 1$, $f_2 = y^2 - 1$, con órden lexicográfico. Si dividimos $f = xy^2 - x$ por $F = (f_1, f_2)$ obtenemos:

$$xy^2 - x = y(xy - 1) + 0(y^2 - 1) + (-x + y)$$

Si tomamos $F = (f_2, f_1)$, obtenemos:

$$xy^2 - x = x(y^2 - 1) + 0(xy - 1) + 0$$

En python:

```
R.<x,y> = PolynomialRing(RR, 2, "xy", order = "lex")
f = x*y^2-x; f1=x*y-1; f2=y^2-1

[q1,q2], r = div_poly([f1,f2],f)
print("Caso 1:")
show(html(f"${f} = {q1} ({f1}) + {q2} ({f2}) + ({r})$"))
print("Caso 2:")
f = x*y^2-x
f1=x*y-1
f2=y^2-1

[q2,q1], r = div_poly([f2,f1],f)
show(html(f"${f} = {q2} ({f2}) + {q1} ({f1}) + {r}$"))
```

Ejemplo del libro

`Caso 1:`

$$x * y^2 - x = x(y^2 - 1.00000000000000) + 0(x * y - 1.00000000000000) + 0$$

`Caso 2:`

$$x * y^2 - x = y(x * y - 1.00000000000000) + 0(y^2 - 1.00000000000000) + (-x + y)$$

Aplicado a polinomios aleatorios

```
R.<x_1,x_2,x_3,x_4> = PolynomialRing(GF(7), order="lex")

f = R.random_element(degree=4, terms = 4)
F = [R.random_element(degree=1, terms=2) for _ in range(3)]
Q, r = div_poly(F, f, R = R)

show(html(f"""
$$
\\begin{{align*}}
f = &{f} \\\\
\\sum_i q_i f_i +r = &{
    sum([q*f for q,f in zip(Q,F)]) + r
    }\\\\
    f_1 = &{F[0]} \\quad \\quad q_1 = {Q[0]} \\\\
    f_2 = &{F[1]} \\quad \\quad q_2 = {Q[1]} \\\\
    f_3 = &{F[2]} \\quad \\quad q_3 = {Q[2]} \\\\
    r = &{r}
\\end{{align*}}
$$
"""))
```

Aplicado a polinomios aleatorios

$$f = 3 * x_1 * x_2 + x_3^2 + 3 * x_3$$

$$\sum_i q_i f_i + r = 3 * x_1 * x_2 + x_3^2 + 3 * x_3$$

$$f_1 = -x_2 + 3 * x_4 \qquad q_1 = -3 * x_1$$

$$f_2 = -2 * x_3 - 3 \qquad q_2 = 3 * x_3 + 1$$

$$f_3 = -x_2 - 1 \qquad q_3 = 0$$

$$r = 2 * x_1 * x_4 + 3$$

Base escalonada

Construcción de la base de buchberger.

```
degree = 3
dim = 3

# Anillo de polinomios que vamos a trabajar
R.<x,y> = PolynomialRing(QQ, order = "lex")

# Ideal de polinomios
I = ideal([R.random_element(degree=10) for _ in range(3)])

# Base de groebner
base = I.groebner_basis()
base
```

Base escalonada

Creando base escalonada

```
# Sorteando la base según el lt de cada polinomio
base = sorted(base, key = lambda x: x.lt())

# Añadiendo el primer elemento
L = [base[0]]

# Añadiendo los polinomios si son 0 módulo los otros elementos
for b in base:
    if b!=R(0) and b.reduce(L) != 0:
        L.append(b)
```

Base escalonada

Intento con elemento en el ideal de la base:

```
f = ideal(L).random_element(degree=3)
Q, r = div_poly(L, f)
```

Tenemos:

$$f = -xy^2 + \frac{1}{8}y^3$$

$$f_1 = y^3 \qquad q_1 = \frac{1}{8}$$

$$f_2 = xy^2 \qquad q_2 = -1$$

$$f_3 = x^4y + 20y^2 \qquad q_3 = 0$$

$$r = 0$$

Base escalonada

Itentando con elemento por fuera del ideal

```
f = R.random_element(degree=3)
Q, r = div_poly(L, f)
```

$$f = \frac{1}{2}x^2y + xy - \frac{1}{2}y^3 + \frac{10}{7}y$$

$$f_1 = y^3 \qquad q_1 = -\frac{1}{2}$$

$$f_2 = xy^2 \qquad q_2 = 0$$

$$f_3 = x^4y + 20y^2 \qquad q_3 = 0$$

$$r = f = \frac{1}{2}x^2y + xy + \frac{10}{7}y$$

Criterio de Buchberger

El algoritmo se basa en el siguiente resultado fundamental (**Cox, Little & O'Shea, §6**):

Teorema 6 (Criterio de Buchberger)
Sea I un ideal polinomial. Entonces una base $G = \{g_1, ..., g_t\}$ de I es una base de Gröbner de I si y solo si para todos los pares $i \neq j$, el residuo de la división de $S(g_i, g_j)$ por G es cero.

Matemáticamente:

$$\forall i \neq j, \quad \overline{S(g_i, g_j)}^{G} = 0$$

Código: Verificación de Base

Esta función es la implementación directa del Teorema 6.

```
def is_Groebner(F):
    """
    Implementación del Teorema 6 (Criterio de Buchberger).
    Retorna True ssi todos los S-polinomios reducen a 0.
    """
    if not F: return True

    # "para todos los pares i != j"
    for i in range(len(F)):
        for j in range(i + 1, len(F)):
            if F[i] == 0 or F[j] == 0: continue

            # Calcular S(gi, gj)
            S_poly = S_polynomial(F[i], F[j])

            # "el residuo de la división por G"
            _, remainder = div_poly(F, S_poly)

            # "es cero"
            if remainder != 0:
                return False # Falló el criterio

    return True
```

Código: S-Polinomio y MCM

Funciones auxiliares para la cancelación de términos líderes (Definición 4 del libro).

```
def S_polynomial(f, g):
    """Calcula S(f, g) = (x^gamma/LT(f))f - (x^gamma/LT(g))g"""
    LT_f = f.lt(); LT_g = g.lt()
    LM_f = f.lm(); LM_g = g.lm()
    LC_f = f.lc(); LC_g = g.lc()

    gamma = lcm_monomials(LM_f, LM_g)
    monom1 = gamma // LM_f
    monom2 = gamma // LM_g

    term1 = (monom1 * f) / LC_f
    term2 = (monom2 * g) / LC_g
    return term1 - term2

def lcm_monomials(m1, m2):
    """MCM componente a componente"""
    exps1 = m1.exponents()[0]
    exps2 = m2.exponents()[0]
    lcm_exps = [max(e1, e2) for e1, e2 in zip(exps1, exps2)]
    return m1.parent().monomial(*lcm_exps)
```

Ejemplos de Prueba

Análisis de ejemplos

Analizamos dos casos contrastantes en $\mathbb{Q}[x, y, z]$ con orden léxico ($x > y > z$):

Caso 1 (Ideal Lineal): $F_1 = \{x + z, y - z\}$
- El S-polinomio es $S(f_1, f_2) = y(x + z) - x(y - z) = xz + yz$.
- Al dividir por F_1, el residuo se cancela totalmente a 0.
- **Conclusión:** Es Base de Gröbner.

Caso 2 (Contraejemplo): $F_2 = \{x^2, xy + y^2\}$
- El S-polinomio es $S(f_1, f_2) = y(x^2) - x(xy + y^2) = -xy^2$.
- Al dividir este término por F_2, obtenemos un residuo no nulo (pues x^2 no divide a xy^2 en orden lex).
- **Conclusión:** NO es Base de Gröbner.

Código de Prueba

=Implementación de los casos en SageMath:

```
R.<x,y,z> = PolynomialRing(QQ, order='lex')

# Caso 1: Una base de Gröbner conocida (debería dar True)
F1 = [x + z, y - z]
print("Probando base de Groebner conocida:")
print(is_Groebner(F1))

# Caso 2: No es base de Gröbner (debería dar False)
F2 = [x^2, x*y + y^2]
print("Probando base que NO es Groebner:")
print(is_Groebner(F2))
```

```
Salida de la ejecución:
Probando base de Groebner conocida:
True
Probando base que NO es Groebner:
False
```

Construcción de la Matriz de Macaulay

Objetivo

Dado un conjunto de polinomios F y un grado D, queremos construir:

- La matriz de Macaulay M
- El conjunto U de monomios ordenados

Cada fila representa un polinomio u·f con grado ≤ D, expresado en la base de monomios U.

Código en Sage

```
def poly_to_Macaulay(F, D):
    R = F[0].parent()
    order = R.term_order()

    # 1. Construir todos los productos u*f con deg(u*f) ≤ D
    UF = []
    for f in F:
        df = f.degree()
        for d in range(D - df + 1):
            for u in R.monomials_of_degree(d):
                UF.append(u * f)

    # 2. Construir el conjunto de monomios que aparecen
    U = set()
    for g in UF:
        for m in g.monomials():
            U.add(m)

    # 3. Ordenar U por el orden monomial del anillo
    U = sorted(U, reverse=True)

    # 4. Construir la matriz de Macaulay
    M = []
```

Código en Sage

```
    for g in UF:
        fila = [g.monomial_coefficient(u) for u in U]
        M.append(fila)

    return Matrix(M), U
```

Reconstrucción de Polinomios desde Macaulay

Objetivo

Dada la matriz de Macaulay en forma escalonada M y el conjunto de monomios U:

- Convertir cada fila no nula en el polinomio que representa
- Normalizar para que el término líder tenga coeficiente 1
- Obtener una base candidata para probar si es Gröbner

Macaulay_to_poly

```
def macaulay_to_Poly(M, U):
    R = U[0].parent()
    polinomios_base = []

    for fila in M.rows():
        # Convierte fila a polinomio
        if not fila.is_zero():
            p = sum(c * m for c, m in zip(fila, U))

            # Normaliza coeficientes líderes
            if p.lc() != 0 and p.lc() != 1:
                 p = p / p.lc()

            polinomios_base.append(p)

    return polinomios_base
```

Experimentación

Código para experimentos

```
def find_Grobner_Basis(F):
    grados = [f.degree() for f in F]
    D = max(grados)
    basis = F

    while is_Groebner(basis) == False:
        M, U = poly_to_Macaulay(F,D)
        M_echelon = M.echelon_form()
        basis = macaulay_to_Poly(M_echelon, U)
        D += 1

    return basis, D-1


R.<x,y,z> = PolynomialRing(GF(7), order='lex')

f1 = x^2 + y^2 +5*x^2*y + x*y*z
f2 = x*y + z
F = [f1, f2]

basis, D = find_Grobner_Basis(F)
for i in range(0, len(basis)):
```

Código para experimentos

```
    print(basis[i])
print(f"D: {D}")

grados = [f.degree() for f in F]
Dmax = max(grados)
print(Dmax)
```

Código para experimentos

```
x^4*y - 2*x*z^3 + 2*y^2*z^2 + y*z^2 - 2*z^4
x^4 - 2*x*z^3 - 2*y^2*z^2 + 2*y*z^2 + 2*z^4 + z^2
x^3*y^2 - y^3*z + y*z^3 + 2*z^3
x^3*y*z - 2*x*z^3 - y^2*z^2 + z^4
x^3*y - 2*y^3*z - y^2*z + 2*y*z^3 - 2*z^3
x^3*z + 2*x*z^3 - 2*y^2*z^2 - y*z^2 + 2*z^4
x^3 + 2*y^3*z - 2*y^2*z - 2*y*z^3 - y*z - 2*z^3
x^2*y^3 - y*z^2
x^2*y^2*z - z^3
x^2*y^2 - z^2
x^2*y*z^2 + x*z^3
x^2*y*z + y^3*z - y*z^3 - 2*z^3
x^2*y + y^3 - y*z^2 - 2*z^2
x^2*z^2 + 2*x*z^3 + y^2*z^2 - z^4
x^2*z + 2*y^3*z + y^2*z - 2*y*z^3 + 2*z^3
x^2 + 2*y^3 + y^2 - 2*y*z^2 + 2*z^2
x*y^4 + y^3*z
x*y^3*z + y^2*z^2
x*y^3 + y^2*z
x*y^2*z^2 + y*z^3
x*y^2*z + y*z^2
x*y^2 + y*z
x*y*z^3 + z^4
x*y*z^2 + z^3
x*y*z + z^2
x*y + z
x*z^2 - y^3*z + y*z^3 + 2*z^3
x*z - y^3 + y*z^2 + 2*z^2
y^4 - y^2*z^2 - 2*y*z^2 + z^2
D: 5
3
```

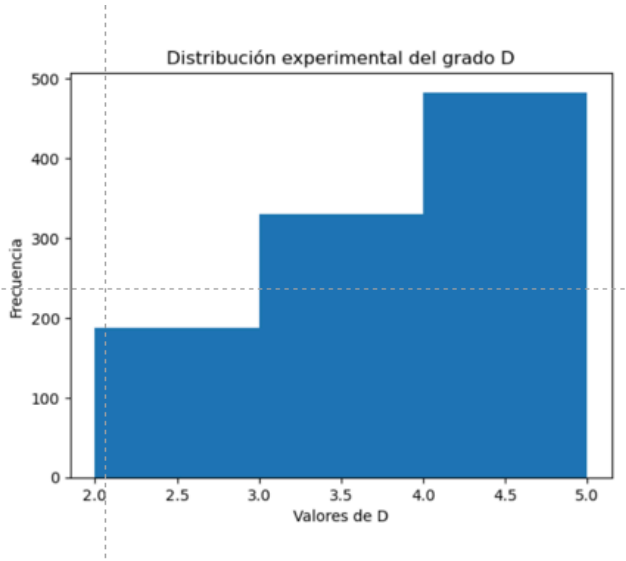
Figure 1: Output de *find_Grobner_Basis* para el input de las diapositivas anteriores

Prueba usando GF(3547)

Frecuencias

```
D
2    187
3    330
4    483
Name: count, dtype: int64
```

```
count    1000.000000
mean        3.296000
std         0.763523
min         2.000000
25%         3.000000
50%         3.000000
75%         4.000000
max         4.000000
Name: D, dtype: float64
```

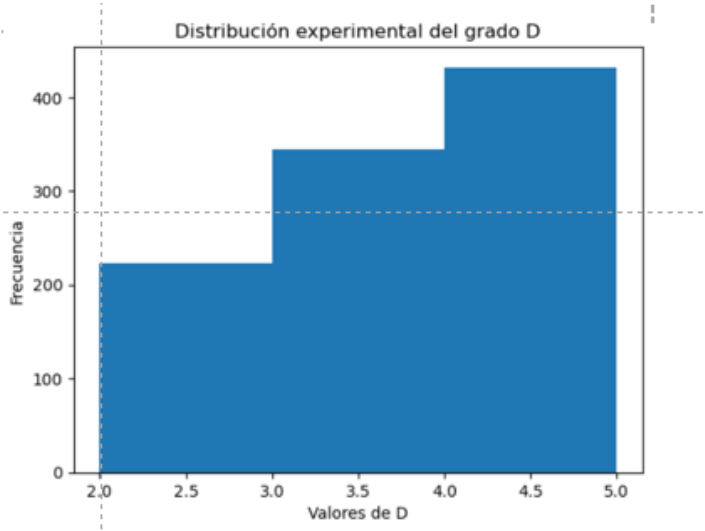


Prueba usando GF(53)

Frecuencias

```
D
2    223
3    345
4    432
Name: count, dtype: int64
```

```
count    1000.00000
mean        3.20900
std         0.78226
min         2.00000
25%         3.00000
50%         3.00000
75%         4.00000
max         4.00000
Name: D, dtype: float64
```

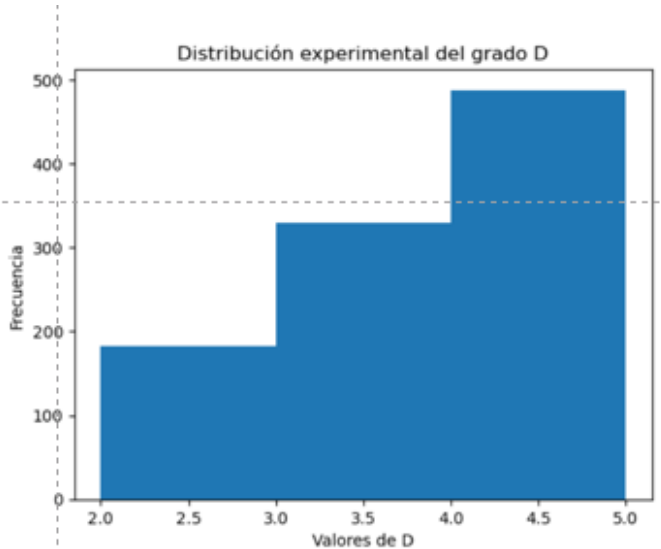


Prueba con GF(673)

Frecuencias

```
D
2    182
3    330
4    488
Name: count, dtype: int64
```

```
count    1000.000000
mean        3.306000
std         0.759566
min         2.000000
25%         3.000000
50%         3.000000
75%         4.000000
max         4.000000
Name: D, dtype: float64
```

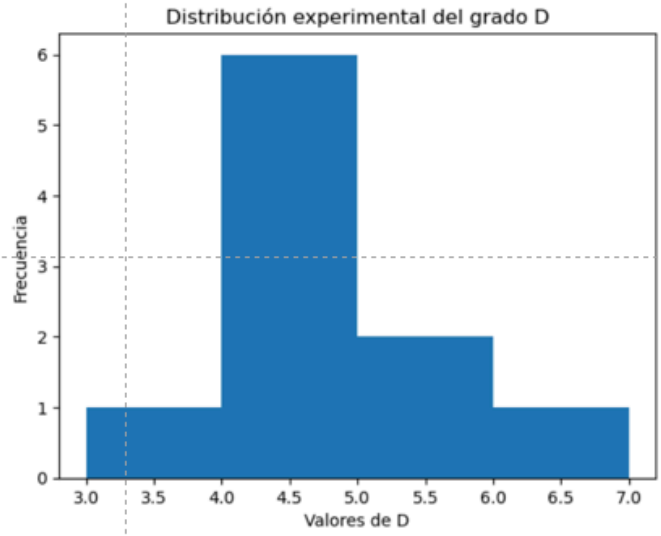


Prueba con cuatro variables para los polinomios cuadráticos

Frecuencias

```
D
3    1
4    6
5    2
6    1
Name: count, dtype: int64
```

```
count    10.000000
mean      4.300000
std       0.823273
min       3.000000
25%       4.000000
50%       4.000000
75%       4.750000
max       6.000000
Name: D, dtype: float64
```



Prueba con un polinomio cuadrático en cinco variables

Frecuencias

```
D
5    1
Name: count, dtype: int64
```

```
count    1.0
mean     5.0
std      NaN
min      5.0
25%      5.0
50%      5.0
75%      5.0
max      5.0
Name: D, dtype: float64
```

Conclusión

El algoritmo en el caso subdeterminado parece bastante eficaz cuando se le trata en un campo finito con tres variables, es capaz de procesar los 1000 ensayos en menos de un minuto y aunque encuentra D-bases de grobner bastante extensas al mirar los resultados parece capaz de encontrar la D-base de grobner en a lo más dos ciclos aunque la moda esta en 4, es decir, el problema suele realizar los dos ciclos pero no más.

Cuando se extiende el número de variables del campo la complejidad parece subir de golpe, ahora un solo ensayo en 4 variables puede tomar más de un minuto, en 10 ensayos se pueden tomar 4 minutos, 10 veces más lo que tomaban 1000 ensayos en 3 variables, en 5 variables un solo ensayo puede tomar de 2 a 4 minutos.

En ninguno de los pocos ensayos realizados en cuatro variables se encontró que el algoritmo tuviese que dar más de cuatro ciclos para encontrar la D-base de grobner, sin embargo, el reducido número de estos no nos permite afirmar nada, igualmente en el caso de 5 variables.