Tarea 4; equipo Padé

Carmen Calderón, Juan Castaño, Daniel Florez, Darío Penagos

Algoritmo Euclidiano Extendido

Pseudocódigo

Input:

- $f, g \in R[x]\ k \in \mathbb{N}$

Pasos:

1. $\rho_0 := \mathrm{lu}(f) \quad r_0 := \mathrm{normal}(f) \quad s_0 := \rho_0^{-1} \quad t_0 := 0$

   $\rho_1 := \mathrm{lu}(g) \quad r_1 := \mathrm{normal}(g) \quad s_1 := 0 \quad t_1 := \rho_0^{-1}$

2. $i := 1$

   `while` $r_i \neq \mathbf{0}$

   $q_i, r_i' := r_{i-1}\ \mathrm{quo\_rem}\ \ r_i$

   $\rho_{i+1} := \mathrm{lu}(r_{i-1} - q_i r_i)$

   $r_{i+1} := (r_{i-1} - q_i r_i)/\rho_{i+1}$

   $s_{i+1} := (s_{i-1} - q_i s_i)/\rho_{i+1}$

   $t_{i+1} := (t_{i-1} - q_i t_i)/\rho_{i+1}$

   $i = i - 1$

   `if` $(\mathrm{degree}(r_i') < k$ `and` $\mathrm{degree}(t_{i+1}) \leq k$ `and` $\mathrm{gcd}(r_{i+1}, t_{i+1}) = 1)$

   `return` $\left(r_i', \frac{t_{i+1}}{\mathrm{constant\_coefficient}(t_{i+1})}, i + 1\right)$

Implementación

1. $\rho_0 := \mathrm{lu}(f)$

   $r_0 := \mathrm{normal}(f)$

   $s_0 := \rho_0^{-1}$

   $t_0 := 0$

   $\rho_1 := \mathrm{lu}(g)$

   $r_1 := \mathrm{normal}(g)$

   $s_1 := 0$

   $t_1 := \rho_0^{-1}$

```
f = R(pol1)
g = R(pol2)
inv_lc1= 1/f.leading_coefficient()
inv_lc2 = 1/g.leading_coefficient()
q = []
i = 1
r = [g*inv_lc2, f*inv_lc1]
rho = [inv_lc2, inv_lc1]
s = [R(1/rho[0]), R(0)]
t = [R(0), R(1/rho[1])]
```

---

2. $q_i, r_i' := r_{i-1}\ \mathrm{quo\_rem}\ \ r_i$

```
qi, ri1 = r[i-1].quo_rem(r[i])
```

Implementación (ii)

3. $\rho_{i+1} := \mathrm{lu}(r_{i-1} - q_i r_i)$

```
if ri1!=0:
    lc = ri1.leading_coefficient()
    inv_lc = 1 / lc
else:
    inv_lc = 1
rho.append(inv_lc)
```

$$r_{i+1} := (r_i')/\rho_{i+1}$$
$$s_{i+1} := (s_{i-1} - q_i s_i)/\rho_{i+1}$$
$$t_{i+1} := (t_{i-1} - q_i t_i)/\rho_{i+1}$$

```
r.append(ri1/rho[i+1])
s.append((s[i-1] - qi*s[i])/rho[i+1])
t.append((t[i-1] - qi*t[i])/rho[i+1])
```

Implementación (iii)

if $(\mathrm{degree}(r'_i) < k$ and $\mathrm{degree}(t_{i+1}) \leq k$ and $\gcd(r_{i+1}, t_{i+1}) = 1)$

return $(r'_i, \frac{t_{i+1}}{\mathrm{constant\_coefficient}(t_{i+1})}, i+1)$

```
if ri1.degree() < k and t[i+1].degree() <= k and r[i+1].gcd(t[i+1]) == 1:
  tj = t[i+1]/(t[i+1].constant_coefficient())

  return ("ok", ri1, tj, i+1)
```

Aproximación de Padé

Pseudocódigo

`Pade_approximation_from_sequence`

Input:

1. $n \in \mathbb{N}$ (donde n es una cota superior del órden de recursión)
2. `seq`, las primeras $2n$ entradas de una sucesión linealmente recurrente en $\mathbb{F}$

Pasos:

1. Construye $f(x) := \sum_{i=0}^{2n-1} a_i x^i$ y $g(x) := x^{2n}$
2. Utiliza `egcd_pade_canonico` para calcular r, t
3. Si $x \mid t$, retorna `falló`
4. $d := \max\{1 + \deg r_j, \deg t_j\}$
5. $m := \mathrm{rev}_d\, t$
6. retorna (`ok`, m)

Implementación

1. Construye $f(x)$ y $g(x)$

```
R = PolynomialRing(F, 'x'); x = R.gen()
f = R(list(seq))
g = x**(2*n)
```

2. Utiliza `egcd_pade_canonico` para calcular r, t

```
res = egcd_pade_canonico(R, f, g, k=n)
if res[0] != "ok":
  return ("no", "eea falló")
_, rj, tj, j = res
```

3. $d := \max\{1 + \deg r_j, \deg t_j\}$

```
d = max(1 + Integer(rj.degree()),
Integer(tj.degree()))
```

Implementación (ii)

4. $m := \mathrm{rev}_d\, t$

```
m = rev_d(tj, d)
```

---

5. retorna (`ok`, m)

```
return ("ok", m)
```

Algoritmo de Wiedemann

Pseudocódigo

`Wiedemann:`

Input:

1. $A \in \mathbb{F}^{n \times n}$
2. $b \in \mathbb{F}^n$

Pasos:

1. Si $b = \vec{\mathbf{0}}$, retornar `1`
2. Escoger un conjunto $U \subset \mathbb{F}$ finito.
3. Escoger $u \in U^n$ de forma uniforme aleatoria, luego computar $u^T A^i b \in \mathbb{F}$ para $0 \leq i \leq 2n$
4. Usar el algoritmo `Pade_approximation_from_sequence` para obtener el mínimo polynomio $m \in \mathbb{F}[x]$ de la suceción linealmente recurrente $\left(u^T A^i b\right)_{i \in \mathbb{N}}$
5. Si $m(A)b = \vec{\mathbf{0}}$, retorna m, en caso contrario, salta a `3`.

Implementación

1. Si $b = \vec{\mathbf{0}}$, retornar 1

```
if not A.is_square():
    raise ValueError("...")
if A.is_singular():
    raise ValueError("...")
```

2. Escoger un conjunto $U \subset \mathbb{F}$ finito.

```
F = A.base_ring()
n = A.nrows()
V = VectorSpace(F, n)
```

3. Escoger $u \in U^n$ de forma uniforme aleatoria, luego computar $u^T A^i b \in \mathbb{F}$ para $0 \leq i \leq 2n$

```
while True:
  u = V.random_element()
  v = b
  seq = [u.dot_product(v)]
  for i in range(1, 2*n):
      v = A * v
      seq.append(u.dot_product(v))
```

Implementación (ii)

4. Usar el algoritmo `Pade_approximation_from_sequence` para obtener el mínimo polynomio $m \in \mathbb{F}[x]$ de la suceción linealmente recurrente $\left(u^T A^i b\right)_{i\in\mathbb{N}}$

```
status, m = Pade_approximation_from_sequence(seq, n, F)
```

---

5. Si $m(A)b = \vec{\mathbf{0}}$, retorna m, en caso contrario, salta a 3.

```
if m(A)*b == vector(F, [0]*n):
  return y
```

Experimento probabilidad

Código

```
N, Q_vals = list(range(5,100)), list(primes(1000))
for n in N:
    probs = []
    for q in Q_vals:
        res = []
        F = Zmod(q)
        U = VectorSpace(F, n)
        for i in range(100):
            while True:
                A = random_matrix(F, n, n)
                b = random_vector(F,n)
                f,_ = A.cyclic_subspace(b, var="x", basis="iterates")
                if f.degree()==n:
                    break
            u = U.random_element()
            res.append(int(f == berlekamp_massey([u*(A^j)*b for j in range(2*n)])))
        probs.append(mean(res))
    with open(f"datos/datos_prob{n}.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["q", "prob"])
        for a,b in zip(Q_vals, probs):
            writer.writerow([a,b])
```

Pseudocódigo

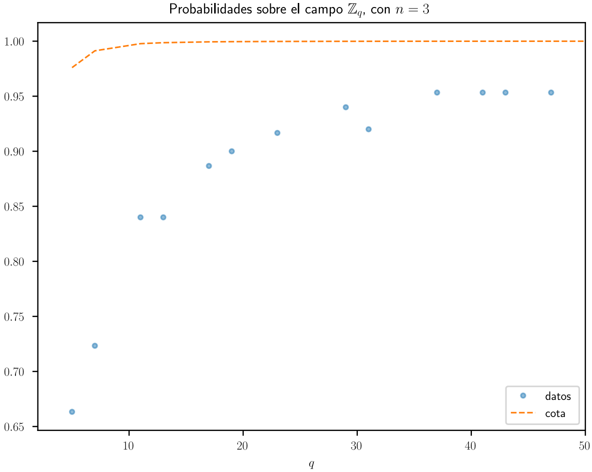
Pasos:

- Escoger una lista de valores para `N` y `Q_vals`
- Para todo `n` ∈ `N`:
  - `probs=[]`
  - Para todo `q` ∈ `Q_vals`:
    - `res=[]`, $\mathbb{F} = \mathbb{Z}_q$
    - Repetir 100 veces:
      - $A \leftarrow \mathbb{F}^{n \times n}$ $b \leftarrow \mathbb{F}^n$
      - $f = \text{min\_poly}\left(\left(A^i b\right)_{i \in \mathbb{N}}\right)$ Si $\text{degree}(f) \neq n$; repetir la escogencia de A y b
      - $u \leftarrow \mathbb{F}^n$
      - `res.append`$\left(f == \text{min\_poly}\left(\left(b^T A^i b\right)_{i \in \mathbb{N}}\right)\right)$
    - `probs.append(mean(res))`
  - `escribir(probs)`

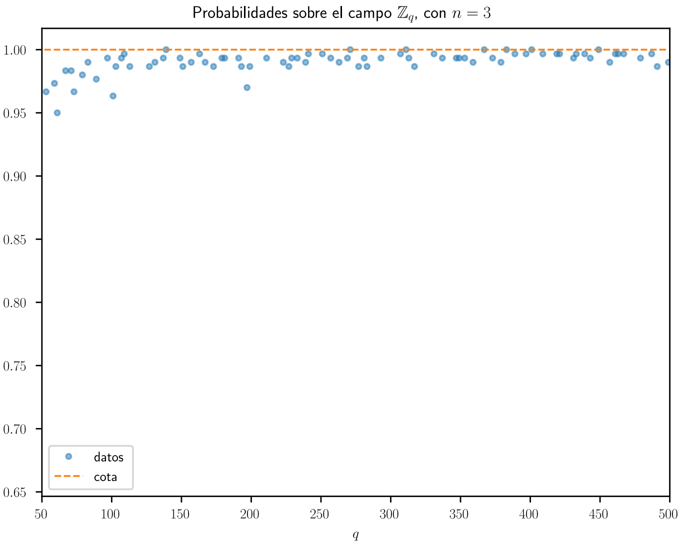
Resultados

Comportamiento respecto n

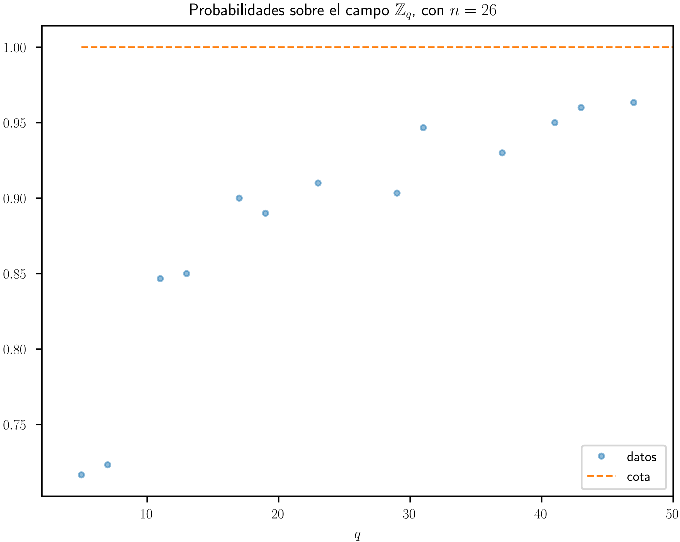
La cota teórica es $p \geq 1 - \frac{d}{\|U\|}$. En general, tomaremos $U = \mathbb{Z}_q^n$. Entonces $p \geq 1 - \frac{n}{q^n}$



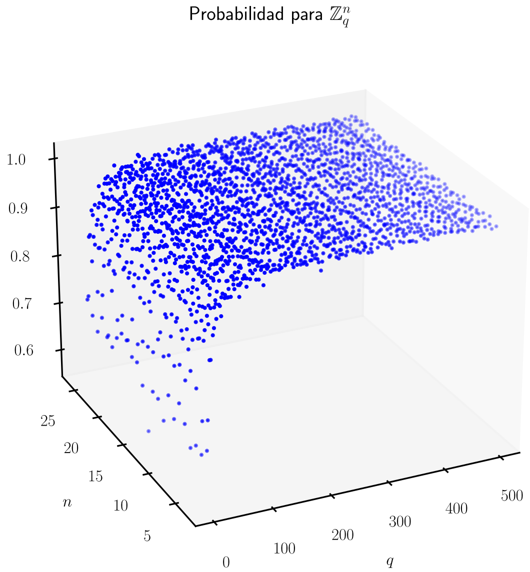
Resultados (ii)



Resultados (iii)



Resultados (iv)



Resultados (v)