Algoritmo para computar la complejidad linear de arreglos periodicos multidimensionales

Lillian González-Albino

Departamento de Ciencia de Cómputos
Universidad de Puerto Rico, Río Piedras

Mayo 20, 2016

1 Resumen del Problema

Moreno y Terkel presentaron una construcción de arreglos periódicos multidimensionales con propiedades de buena correlación y complejidad. Para analizar la complejidad de estos arreglos se "desenvolvían" utilizando el Teorema del Residuo Chino y luego se analizaban con el algoritmo Berlekamp-Massey. Pero este método presentaba una restricción en las dimensiones del arreglo pues tenían que ser coprimos. En [4, 5] fue propuesto una teoría nueva para analizar la complejidad linear de arreglos multidimensionales que no tiene esta restricción; provee una definición de complejidad linear de arreglos multidimensionales que es consistente con la definición de complejidad linear en una dimensión. En esta investigación se implementa el algoritmo desarrollado en [3] sobre la teoría presentada en [4, 5].

2 Trabajo realizado

Se construyó un arreglo periódico de dos dimensiones A con construcción Moreno-Tirkel y, utilizando el programa, se implementó el algoritmo Rubio-Sweedler (respecto al grado lexicográfico $x > y$) para obtener el ideal de **polinomios válidos** en A, definido como $Val(A)$, para calcular su **complejidad lineal**.

2.1 Definiciones

Definición 1: Sea $\alpha \in N_0^m$. Un polinomio $C(x) = \sum c_i x^i$ es **valido en el arreglo** a si $\forall\ \beta \in N_0^m$

$$\sum c_\alpha s_{\alpha+\beta} = 0$$

Lema 1: Sean Δ, Γ_0. Los siguientes son equivalentes:
1. Para $\beta \in \Delta, \alpha \in N_0$, si $\alpha \leq \beta$ entonces $\alpha \in \Delta$.
2. Para $\alpha \in \Gamma, \beta \in N_0$, si $\alpha \leq \beta$ entonces $\beta \in \Gamma$.[2]

El conjunto de exponentes de los monomios que no sean monomios líderes en el ideal forman el **conjunto delta** ($\Delta_I = N_0^n \setminus LE(I)$). Donde $LE(I)$ se refiere al exponente del monomio líder del ideal I. [2]

Definición 2: Sea A un arreglo periódico multidimensional y $Val(A)$ el ideal de polinomios válidos en él, se define la **complejidad lineal** de A como la cantidad de elementos en el conjunto $\Delta_{Val(A)}$. Esto es, $\mathcal{L}(A) = |\Delta_{Val(A)}|$ [7].

Definición 3: Sea A un arreglo periódico 2-dimensional y (n_1, n_2) su vector de periodo. Se define la **complejidad lineal normalizada** $\mathcal{L}_n(A)$ del arreglo A como $\mathcal{L}_n(A) = \mathcal{L}(A)/n_1 n_2$ [2].

2.2 Complejidad del arreglo A con construcción Moreno-Tirkel

Sea $s_i = (0, 1, 1, 0, 1, 0, 0, ...)$ una sucesión con buena complejidad lineal y $t_i = 1, 3, 2, 6, 4, 5$ es una sucesión con buena correlación, entonces $(a_{i,j})$ se define por $a_{i,j} = s_{jt_i}$.

0	0	1	0	1	1
1	1	0	0	1	0
0	1	1	0	0	0
1	0	1	1	0	0
1	0	0	0	0	1
0	0	0	1	1	0
0	1	0	1	0	1

Table 1: Arreglo A = $a_{i,j}$

Los polinomios válidos en el arreglo A son $Val(A) = \{y^4 + y^3 + y^2 + 1,\ y^3x + y^3 + yx + x + y + 1,\ x^6 - 1\}$. Entonces podemos calcular que

$\Delta_{Val(A)} = \{1, y, x, y^2, xy, x^2, y^3, xy^2, x^2y, x^3, x^2y^2, x^3y, x^4, x^3y^2, x^4y, x^5, x^4y^2, x^5y, x^5y^2\}$

y que $\mathcal{L}(A) = |\Delta_{Val(A)}| = 19$

La complejidad normalizada del arreglo es:

$$\mathcal{L}_n(A) = \frac{19}{42} \approx \ 0.45.$$

3 Algoritmo del programa

Algoritmo para encontrar el polinomio válido en la secuencia s

Input: secuencia s, módulo m
Output: polinomio válido para la secuencia s
Initialization
$PrimeraFila = s + s$
$u = 2 *$ (largo de s) $- 1$
create lista $Mult$ de pares ordenados (0,y) en orden monomial grlex
create matriz $Tabla$ con $PrimeraFila$ como única fila
delete primer elemento de $Mult$
while hayan elementos en $Mult$ **do**
    delete primer elemento de $PrimeraFila$
     $PróximaFila = PrimeraFila$
     $u = u - 1$
    append $PróximaFila$ a $Tabla$ como fila abajo
     $l =$ cantidad de filas en $Tabla$
    delete primer elemento de $Mult$
    delete columnas (por la derecha) de $Tabla$ hasta que tengas u cantidad de columnas
    create lista $Coef$ de coeficientes
    for cada combinación y (y vector) de números desde 0 hasta m y de largo l **do**
        $vect = (\ \ Tabla * y\ \ )\ \ \%\ \ m$
        if $vect$ es un vector cero **then**
            append $vect$ a $Coef$
    delete primer elemneto de $Coef$
    if $Coef$ no es vacío **then**
        delete todos los elementos de $Mult$
    save $vect$ como los coeficientes del polinomio valido en s y con variables correspondintes 1 hasta (0,l)

Algoritmo para encontrar polinomios de recurrencia válidos para el arreglo periódico de dos dimensiones A **con construcción Moreno-Tirkel**
Input: secuencia s, secuencia t, módulo m

Output: polinomios válidos para el arreglo A
Initialization
create matriz A como arreglo periodico 2D con construcción Moreno-Tirkel
$u = 2 * (\text{ largo de } s + \text{ largo de } t\ ) - 2$
create matriz M con dimensiones uxu con M repetida periódicamente
create lista $Tracker$
create lista $Mult$ de pares ordenados (x_0, y_0) en orden monomial grlex
create matriz $Tabla$
$PrimeraFila$ = lista de elementos en M cuando se recorre diagonalmente de arriba hacia abajo, izquierda a derecha
x = primer elemento de $Mult$, x_0
y = primer elemento de $Mult$, y_0
$u = u - (x + y)$
append $PrimeraFila$ a $Tabla$ como primera fila
delete primer elemento en $Mult$
while hayan elementos en $Mult$ **do**
    $temp$ = copia de M
    x = primer elemento de $Mult$, x_0
    y = primer elemento de $Mult$, y_0
    delete x cantidad de columnas (por la izquierda) de $temp$
    delete y cantidad de filas (por abajo) de $temp$
    $PróximaFila$ = lista de elementos en $temp$ cuando se recorre diagonalmente de arriba hacia abajo, izquierda a derecha
    $u = u - (x + y)$
    append $PróximaFila$ a $Tabla$ como fila abajo
    l = cantidad de filas en $Tabla$
    append primer elemento de $Mult$ a $Tracker$
    delete primer elemento de $Mult$
    delete columnas de $Tabla$ (por la izquierda) hasta que tengas u cantidad de columnas
    create lista $Coef$ para coeficientes
    for cada combinación y (y vector) de números 0 hasta m, y de largo l **do**
        $vect = (\ Tabla * y\ )\ \%\ m$
        if $vect$ es un vector cero **do**
            append $vect$ a $Coef$
    delete primer elemento de $Coef$
    if $Coef$ no es vacío **do**
        for cada elemento i en $Coef$ **do**
            if primer elemento en $i = 1$ **do**
                $elemento = Coef[i]$
    if $elemento$ existe **do**
        $coeficientes = k$
        $variables = Tracker$
        save $coeficientes$ y $variables$ como polinomio válido en A
        delete última fila de $Tabla$
        delete múltiplos del último elemento de $Tracker$ de la lista $Mult$

delete último elemento de *Tracker*

4 Trabajo pendiente

- Se trató de correr el programa con un arreglo de periodo 11 y se notó un aumento significativo en el tiempo corriendo, está pendiente optimizar el programa para que pueda procesar arreglos más grandes en menos tiempo.
- Crear una interface donde se pueda accesar el programa.
- Comparar la complejidad de las construcciones de Moreno Tirkel con otras construcciones conocidas como multisucesión.

5 Apéndice

5.1 Código

```
1
2  #Input: dos listas del mismo largo una contiene coeficientes, otra
3  #los grados de un monomio xy
4  #Output: void
5  #A ade polinomio (como string) a  una variable global
6  def PolinomioValido( vectorCoeff, rowTracker ):
7      #rowTracker es una lista de pares ordenados
8      #numero de elementos en vector debe ser igual al numero de
9      #elementos en la lista rowTracker
10     PoVa = ""
11     vLen = len(vectorCoeff)
12     rtLen = len(rowTracker)
13     P = []
14     if(vLen == rtLen):
15         for i in range(vLen):
16             x = rowTracker[i]
17             y = vectorCoeff[i]
18             if(y != 0):
19                 z = (y, x)
20                 P.append(z)
21     #P es una lista de monomios (polinomio valido)
22     rn = len(P)
23     strn = ""
24     for i in range(rn, 0, -1):
25         k = P[i - 1]
26         #k es de forma (z, (x,y)) monomio
27         #k[0] es el coeficiente
28         l = k[1]
29         #l es de forma (x,y)
30
31         #(0,0)
32         if(l[0] == 0 and l[1] == 0):
33             #1 * (0,0)
34             if(k[0] == 1):
```

```
                strn = strn + " 1 "
            # c * (0,0)
            if(k[0] != 1 and k[0] != 0):
                strn = strn + " %d " %(k[0])
        #(0,y)
        if(l[0] == 0 and l[1] != 0):
            # (0,1)
            if(l[1] == 1):
                # 1 * (0,1)
                if(k[0] == 1):
                    strn = strn + " y +"
                # c * (0,1)
                if(k[0] != 1 and k[0] != 0):
                    strn = strn + " %d y +" %(k[0])
            #(0,y)
            if(l[1] != 1):
                # 1 * (0,y)
                if(k[0] == 1):
                    strn = strn + " y^%d +" %(l[1])
                # c * (0,y)
                if(k[0] != 1 and k[0] != 0):
                    strn = strn + " %d y^%d +" %(k[0],l[1])
        #(x,0)
        if(l[0] != 0 and l[1] == 0):
            #(1,0)
            if(l[0] == 1):
                # 1 * (1,0)
                if(k[0] == 1):
                    strn = strn + " x +"
                # c * (1,0)
                if(k[0] != 1 and k[0] != 0):
                    strn = strn + " %d x +" %(k[0])
            #(x,0)
            if(l[0] != 1):
                # 1 * (x,0)
                if(k[0] == 1):
                    strn = strn + " x^%d +" %(l[0])
                # c * (x,0)
                if(k[0] != 1 and k[0] != 0):
                    strn = strn + " %d x^%d +" %(k[0],l[0])
        #(x,y)
        if(l[0] != 0 and l[1] != 0):
            #(x,1)
            if(l[0] != 1 and l[1] == 1):
                # 1 * (x,1)
                if(k[0] == 1):
                    strn = strn + " y x^%d +" %(l[0])
                # c * (x,1)
                if(k[0] != 1 and k[1] != 0):
                    strn = strn + " %d y x^%d +" %(k[0],l[0])
            #(1,y)
            if(l[0] == 1 and l[1] != 1):
                # 1 * (1,y)
                if(k[0] == 1):
                    strn = strn + " y^%d x +" %(l[1])
                # c * (1,y)
                if(k[0] != 1 and k[0] != 0):
```

```
                    strn = strn + " %d y^%d x +" %(k[0],l[1])
            #(x,y)
            if(l[1] != 1 and l[0] != 1):
                # 1 * (x,y)
                if(k[0] == 1):
                    strn = strn + " y^%d x^%d +" %(l[1], l[0])
                # c * (x,y)
                if(k[0] != 1 and k[0] != 0):
                    strn = strn + " %d y^%d x^%d +" %(k[0],l[1], l
    [0])
            #(1,1)
            if(l[0] == 1 and l[1] == 1):
                # 1 * (1,1)
                if(k[0] == 1):
                    strn = strn + " y x +"
                # c * (1,1)
                if(k[0] != 1 and k[0] != 0):
                    strn = strn + " %d y x +" %(k[0])
    PoVa = PoVa + strn
    #ahora tengo el polinomio como string
    slen = len(PoVa)
    if(PoVa[slen - 1] == "+"):
        sLen = len(PoVa) - 1
        PoVa = PoVa [0:sLen]
    #escribo el polinomio en documento
    f.write("%s \n" %(PoVa))
    global PV
    PV.append(PoVa)

#Input: dos listas del mismo largo una contiene coeficientes,
#otra los grados de un monomio xy
#Output: polinomio valido en la secuencia como string
def PolinomioValidoSecuencia( vectorCoeff, rowTracker ):
    #rowTracker es una lista de pares ordenados
    #numero de elementos en vector debe ser igual al
    #numero de elementos en la lista rowTracker
    PoVa = ""
    vLen = len(vectorCoeff)
    rtLen = len(rowTracker)
    P = []
    if(vLen == rtLen):
        for i in range(vLen):
            x = rowTracker[i]
            y = vectorCoeff[i]
            if(y != 0):
                z = (y, x)
                P.append(z)
    #P es una lista de monomios (polinomio valido)
    rn = len(P)
    strn = ""
    for i in range(rn, 0, -1):
        k = P[i - 1]
        #k es de forma (z, (x,y)) monomio
        #k[0] es el coeficiente
        l = k[1]
        #l es de forma (x,y)
        #(0,0)
```

```
148         if(l[0] == 0 and l[1] == 0):
149             #1 * (0,0)
150             if(k[0] == 1):
151                 strn = strn + " 1 "
152             # c * (0,0)
153             if(k[0] != 1 and k[0] != 0):
154                 strn = strn + " %d " %(k[0])
155         #(0,y)
156         if(l[0] == 0 and l[1] != 0):
157             # (0,1)
158             if(l[1] == 1):
159                 # 1 * (0,1)
160                 if(k[0] == 1):
161                     strn = strn + " y +"
162                 # c * (0,1)
163                 if(k[0] != 1 and k[0] != 0):
164                     strn = strn + " %d y +" %(k[0])
165             #(0,y)
166             if(l[1] != 1):
167                 # 1 * (0,y)
168                 if(k[0] == 1):
169                     strn = strn + " y^%d +" %(l[1])
170                 # c * (0,y)
171                 if(k[0] != 1 and k[0] != 0):
172                     strn = strn + " %d y^%d +" %(k[0],l[1])
173     PoVa = PoVa + strn
174     #ahora tengo el polinomio como string
175     slen = len(PoVa)
176     if(PoVa[slen - 1] == "+"):
177         sLen = len(PoVa) - 1
178         PoVa = PoVa [0:sLen]
179     return PoVa
180 
181 #Input: matrix cuadrada "matriz" y numero m de columnas de la
182 #matrix "matriz"
183 #Output: matriz rotada hacia la derecha
184 #se rota la matriz para poder recorrerla diagonalmente
185 #(ya que los indices son de izq. a der. pero de arriba hacia
186 #abajo)
187 def rotateMatrixRight(matriz, m):
188     temp = matrix(ZZ, m)
189     fila = 0
190     cols = matriz.ncols() - 1
191     matrizCols = matriz.ncols()
192     for fila in range(matrizCols):
193         element = 0
194         for element in range(matrizCols):
195             temp[element, cols] = matriz[fila, element]
196         cols = cols - 1
197     return temp
198 
199 #Input: matriz mat
200 #Output: matriz nueva con dimensiones iguales (cudrada)
201 #se hace cuadrada para poder recorrer la matriz en la funcion
202 #recorreMatriz
203 def makeMatrixSquare(mat):
204     temp = copy(mat)
```

```
205     rows = temp.nrows()
206     cols = temp.ncols()
207     #si tiene mas columnas, borra columnas
208     if( rows < cols ):
209         while( rows < cols ):
210             temp = temp.delete_columns([cols-1])
211             cols = temp.ncols()
212             rows = temp.nrows()
213     #si tiene mas filas, borra filas
214     else:
215         while( cols < rows ):
216             temp = temp.delete_rows([0])
217             rows = temp.nrows()
218             cols = temp.ncols()
219     return temp
220 
221 #Input: matrix mat y int u
222 #Output: lista de elementos cuando recorres la matriz
223 #diagonalmente de izq a derecha,
224 #arriba hacia abajo
225 def recorreMatriz(mat, u):
226     temp = copy(mat)
227     temp = makeMatrixSquare(temp)
228     m = temp.ncols()
229     #indices de la matriz de izq->der, arr->abajo.
230     #rotate para recorrer matriz
231     temp = rotateMatrixRight(temp, m)
232     w = temp.ncols()
233     grado = 0
234     fila = []
235     for grado in range(w): #u
236         x = 0
237         for x in range(grado+1):
238             y = grado - x
239             value = temp[x,y]
240             fila.append(value)
241     return fila
242 
243 #Input: par ordenado (x,y) po (int), matriz mat
244 #Output: matrix nueva con shift po
245 def makeShift(po, mat):
246     temp = copy(mat)
247     x = po[0]
248     y = po[1]
249     i = 0
250     for i in range(x):
251         temp = temp.delete_columns([0])
252     j = 0
253     for j in range(y):
254         matRows = temp.nrows() - 1
255         temp = temp.delete_rows([matRows])
256     return temp
257 
258 #Input: matriz mat, lista de pares ordenados, int u
259 #Output: lista de elementos de la matriz mat con shift del
260 #primer elemento
261 #en la lista listM y calcula el proximo u (para la proxima fila)
```

```
262 def add_next_row(mat, listaM, m):
263     parOrd = listaM[0]
264     temp = makeShift(parOrd, mat)
265     x = parOrd[0]
266     y = parOrd[1]
267     #calcular valor de u
268     u = m - (x + y)
269     fila = recorreMatriz(temp, u)
270     return fila, u
271 
272 #Input: matriz mat, int u
273 #Output: reduce matriz mat a u cantidad de columnas
274 def reduceMatrixColumnsToU(mat, u):
275     temp = copy(mat)
276     rows = temp.nrows()
277     cols = temp.ncols()
278     if( u < cols ):
279         while( u < cols ):
280             temp = temp.delete_columns([cols-1])
281             cols = temp.ncols()
282     return temp
283 
284 #Input: int gradTot (grado total)
285 #Output: lista de pares ordenados (x,y) en orden grlex de
286 #monomios
287 def listaMultiplicadores(gradTot):
288     A = []
289     GT = 0
290     for GT in range(gradTot+1):
291         x = 0
292         for x in range(GT + 1):
293             y = GT - x
294             P = x ,y
295             A.append(P)
296     return A
297 
298 #Input: int u (grado total)
299 #Output: lista de pares ordenados (0,y) en orden grlex de
300 #monomios
301 def listaMultiplicadoresSecuencia(u):
302     lista = []
303     for i in range(u + 1):
304         elemento = 0, i
305         lista.append(elemento)
306     return lista
307 
308 #Input: dos matrices m1, m2 con misma cantidad de columnas
309 #Output: una matriz conpuesta de m2 encima de la matriz m1
310 def matrixAppendTop(m1,m2):
311     if( m1.ncols() != m2.ncols() ):
312         return "must have the same number of columns"
313     else:
314         bigMrows = m1.nrows() + m2.nrows()
315         m1cols = m1.ncols()
316         bigM = matrix(ZZ, bigMrows, m1cols)
317         i = bigMrows - 1
318         for i in range(i, -1, -1):
```

```
            j = 0
            for j in range(m1cols):
                m1rows = m1.nrows()
                w = bigMrows - m1rows
                if(i >= w):
                    k = i - m1rows
                    bigM[i,j] = m1[k,j]
                else:
                    bigM[i,j] = m2[i,j]
    return bigM

#Input: una matriz "matriz" (periodica)y int m (u)
#Output: matriz periodica con u columnas y u filas(repite matriz)
def enlargeMatrix(matriz, m):
    temp = matriz
    tempCols = temp.ncols()
    tempRows = temp.nrows()
    while(tempCols < m):
        temp = temp.augment(temp)
        tempCols = temp.ncols()
    while(tempRows < m):
        temp = matrixAppendTop(temp, temp)
        tempRows = temp.nrows()
    while(tempCols > m):
        temp = temp.delete_columns([tempCols -1])
        tempCols = temp.ncols()
    while(tempRows > m):
        temp = temp.delete_rows([0])
        tempRows = temp.nrows()
    return temp

#Input: par ordenado int (x,y) parOrd, par ordenado int (x,y)
#multiplo
#Output: true or false, si multiplo es multiplo de parOrd
#(como degrees de monomios en dos variables)
def es_multipo(parOrd, multiplo):
    #parOrd(0,0)
    if(parOrd[0] == 0 and parOrd[1] == 0):
        return false
    #parOrd(x,0)
    if(parOrd[0] != 0 and parOrd[1] == 0):
        #mult(0,0) o (0,y)
        if(multiplo[0] == 0):
            return false
        #mult(x,0) o (x,y)
        if(multiplo[0] != 0):
            if(parOrd[0] <= multiplo[0]):
                return true
    #parOrd(0,y)
    if(parOrd[0] == 0 and parOrd[1] != 0):
        #mult(0,0) o (x,0)
        if(multiplo[1] == 0):
            return false
        #mult(0,y) o (x,y)
        if(multiplo[1] != 0):
            if(parOrd[1] <= multiplo[1]):
                return true
```

```
376     #(x,y)
377     if(parOrd[0] != 0 and parOrd[1] != 0):
378         #mult(0,0), (x,0), (0,y)
379         if(multiplo[0] == 0 or multiplo[1] == 0):
380             return false
381         #mult(x, y)
382         if((parOrd[0] <= multiplo[0]) and (parOrd[1] <= multiplo
    [1])):
383             return true
384
385 #Input: dos listas
386 #Output: diferencia de dos listas (conserva orden de elementos)
387 diff = lambda l1,l2: [x for x in l1 if x not in l2]
388
389 #Input: lista "lista" de pares ordenados (x,y) int, par ordenado
390 #(x,y) po int
391 #Output: lista de "lista" sin multiplos de po (como degrees de
392 #monomios en dos variables)
393 def delete_multiplo_off_list(lista, po):
394     ran = len(lista)
395     elements = []
396     i = 0
397     for i in range(ran):
398         mult = lista[i]
399         if(es_multipo(po, mult)):
400             elements.append(lista[i])
401     lista2 = diff(lista, elements)
402     return lista2
403
404 #Input: matriz mat, int modu
405 #Output:devuelve una matriz nueva = mat % modu
406 def matrix_in_module(mat, modu):
407     temp = copy(mat)
408     rows = mat.nrows()
409     cols = mat.ncols()
410     i = 0
411     for i in range(rows):
412         j = 0
413         for j in range(cols):
414             temp[i,j] = mat[i,j] % modu
415     return temp
416
417 #Input: matriz mat, lista vec
418 #Output: a ade fila vec a la matriz (ultima fila) si la cantidad
419 #de elementos es menos que la cantidad de columnas, se le a ade
420 #-1 en los espacios "vacios"
421 def append_row_bottom(mat, vec):
422     vecLength = len(vec)
423     tRows = mat.nrows() + 1
424     tCols = mat.ncols()
425     temp = matrix(ZZ, tRows, tCols)
426     i = 0
427     rows = mat.nrows()
428     if(tCols == vecLength):
429         for i in range(tRows):
430             if(i < rows):
431                 j = 0
```

```
                for j in range(tCols):
                    temp[i,j] = mat[i,j]
            else:
                j = 0
                for j in range(tCols):
                    temp[i,j] = vec[j]
        return temp
    if(tCols > vecLength):
        for i in range(tRows):
            j = 0
            if(i < rows):
                for j in range(tCols):
                    temp[i,j] = mat[i,j]
            else:
                for j in range(tCols):
                    if(j < vecLength):
                        temp[i,j] = vec[j]
                    else:
                        temp[i,j] = -1
        return temp
    if(vecLength > tCols):
        print("vector elements exceed amount of columns in matrix")

#Input: una lista sequence (periodic sequence), una lista shifts
#Output: una matriz con construccion Moreno-Tirkel (secuencia
#hacia arriba)
def makeMatrixShiftUp(sequence, shifts):
    listaCols = []
    tLen = len(shifts)
    sLen = len(sequence)
    #cantidad de columnas
    for i in range(tLen):
        #el shift
        elShift = int(shifts[i])
        #columna vacia
        columna = []
        for j in range(sLen):
            columna.append(sequence[(j - elShift) % sLen])
        #columna esta alrevez - flip columna
        cLen = len(columna)
        columna2 = []
        for k in range(cLen):
            columna2.append(columna[cLen - k - 1])
        listaCols.append(columna2)
    matriz = column_matrix(ZZ, listaCols)
    return matriz

#Input: una lista sequence (periodic sequence), una lista shifts
#Output: una matriz con construccion Moreno-Tirkel (secuencia
#hacia abajo)
def makeMatrixShiftDown(sequence, shifts):
    listaCols = []
    tLen = len(shifts)
    sLen = len(sequence)
    #cantidad de columnas
    for i in range(tLen):
        #el shift
```

```
        elShift = int(shifts[i] + 1)
        #columna vacia
        columna = []
        for j in range(sLen):
            columna.append(sequence[(j + elShift) % sLen])
        listaCols.append(columna)
    matriz = column_matrix(ZZ, listaCols)
    return matriz


#start of program
from itertools import product
from time import gmtime, strftime

#abrir archivo y guardar contenido previo del archivo
#note: esto es para cuando corre en sage, se tiene que cambiar
#el path de la funcion open al path del archivo en la computadora
#que corra el programa
f=open('/Users/Lillian/Desktop/mydat2.txt', 'r+')
f.read()

#variable global: lista para guardar polinomios validos en la
#secuencia
PV = []

#marca el comienzo de la sesion
f.write("Start of session. Time: ")
f.write(strftime("%Y-%m-%d %H:%M:%S\n", gmtime()))

#intreact: input de la secuancia s y los shifts t
@interact
def _listas_(seq = input_box(label = 'sequence s = ', default =
    [0,1,1,0,1,0,0], type = list, width=30), shi = input_box(label
    = 'shifts t = ', default = [1,3,2,6,4,5], type = list, width
    =30), modu = input_box(label = 'module = ',default= 2, type=int
    , width=30)):

    #vaciar lista de polinomios validos
    while len(PV) > 0 : PV.pop()

    #escribe en el archivo la secuencia y los shifts
    f.write(" sequence s: %s \n shifts   t: %s \n" %(str(seq),str(
    shi)))

    #*** buscar polinomio de recurrencia valido en la secuencia
    # fila 1 : sequence dos veces (secuencia periodica)
    firstSRow = seq + seq

    # u = 2 ( length of seq ) - (dimensiones)
    # u se llama lim
    lim = 2 * (len(seq)) - 1

    #lista de monomios en grlex una variable
    #(0,0),(0,1),(0,2),   ,(0,u)
    listaMultSec = listaMultiplicadoresSecuencia(lim)

    #crear tabla (matriz)
```

```
    Scols = len(listaMultSec)
    STable = matrix(ZZ, 1, Scols)

    #adjuntar fila 1 a la tabla
    STable = append_row_bottom(STable, firstSRow)
    STable = STable.delete_rows([0])
    del listaMultSec[0]
    StableRows = STable.nrows()

    #a adir filas y buscar dependencia linear
    while((listaMultSec)):

        # calcular proxima fila
        #(proxima fila = fila1 - primer elemento)
        # cuando a ades proxima fila nextSRos elimina
        #primer elemento de firstSRow
        del firstSRow[0]
        nextSRow = firstSRow

        #actualizar u (u se llama lim)
        lim = lim - 1

        #adjuntar fila nueva a la tabla
        STable = append_row_bottom(STable, nextSRow)
        StableRows = STable.nrows()

        #eliminar primer elemento de listaMultSec
        #(actualizar condicion del while)
        del listaMultSec[0]

        #matriz debe ser reducida (columna wise) hasta u(lim)
        #para que solo sume hasta donde sea necesario
        SMT = reduceMatrixColumnsToU(STable, lim)
        filaCheck =  STable.row(-1)
        if(filaCheck[0] == -1):
            break

        #lista de coeficientes (por fila)
        SlistaCoeff = []

        #range modu (modulo) y "length" del coef es el numero
        #de filas
        for y in product(range(modu) , repeat= StableRows):
            vec = vector(y)
            smultiplied = ( vec * SMT ) % modu

            #guarda todas las combinaciones de coef que devuelven
            #vector 0 (dependencia linear) van a haber mas de uno
            #porque el vector 0 devuelve 0 tambien
            if(smultiplied == 0):
                SlistaCoeff.append(vec)

        #elimina elemento cero (0,0,0,...,0_mod)
        del SlistaCoeff[0]

        #if SlistaCoeff (si encontre coeficientes)
        if((SlistaCoeff)):
```

```
598             #salir del while
599             while len(listaMultSec) > 0 : listaMultSec.pop()
600 
601     #polinomio recurrencia val en la secuencia
602     lim = len(SlistaCoeff[0]) - 1
603     SMultList = listaMultiplicadoresSecuencia(lim)
604     Polinomio = PolinomioValidoSecuencia( SlistaCoeff[0], SMultList
        )
605 
606     #display polinomio
607     show("polinomio valido en la secuencia = ", Polinomio)
608 
609     #guarda polinomio valido en la secuencia en el archivo
610     f.write("\n %s      Val(s)\n\n\t A = \n" %(Polinomio))
611 
612     #***termina busqueda de polinomio de recurrencia en la
613     #***secuencia
614 
615     #***buscar polinomios validos en el arreglo multidimensional
616 
617     #crear y desplegar matriz (array) con construccion
618     #Moreno-Tirkel
619     p = makeMatrixShiftUp(seq, shi)
620     n1 = len(seq)
621     n2 = len(shi)
622     show(p)
623 
624     #agrandar matriz (para recorrer)
625     p = matrix_in_module(p, modu)
626     m = 2*(n1 + n2) - 2
627     matrizGrande = enlargeMatrix(p, m)
628 
629     #guardar matrix al archivo
630     r = p.nrows()
631     c = p.ncols()
632     for i in range(r):
633         strn = " [ "
634         for j in range(c):
635             strn = strn + " %d " %(p[i,j])
636         strn = strn + " ]\n"
637         f.write(strn)
638     f.write("\n     ")
639 
640     #trackers de filas
641     MultRowTracker = []
642     MultList = listaMultiplicadores(m-1)
643     MultRowTracker.append(MultList[0])
644 
645     #crear tabla Table
646     cols = len(MultList)
647     Table = matrix(ZZ, 1, cols)
648 
649     #a adir primera fila
650     rowOne, u = add_next_row(matrizGrande, MultList, m)
651     Table = append_row_bottom(Table, rowOne)
652     Table = Table.delete_rows([0])
653     del MultList[0]
```

```
654     tableRows = Table.nrows()
655 
656     #a adir filas y buscar dependencia linear
657     while((MultList)):
658 
659         #calcular proxima fila
660         nextRow, u = add_next_row(matrizGrande, MultList, m)
661 
662         #adjuntar proxima fila a Table
663         Table = append_row_bottom(Table, nextRow)
664         tableRows = Table.nrows()
665 
666         #actualizar trackers
667         MultRowTracker.append(MultList[0])
668         del MultList[0]
669 
670         #matriz debe ser reducida (columna wise) hasta u
671         #para que solo sume hasta donde sea necesario
672         w = ( u * (u + 1 ) ) / 2
673         MT = reduceMatrixColumnsToU(Table, w)
674         filaCheck =  Table.row(-1)
675         if(filaCheck[0] == -1):
676             break
677 
678         #BUSCAR DEPENDENCIA LINEAR
679 
680         #lista de coeficientes (por fila)
681         listaCoeff = []
682 
683         #range modu (modulo) y "length" de coef es el numero
684         #de filas
685         for x in product(range(modu) , repeat= tableRows):
686             vec = vector(x)
687             multiplied = ( vec * MT ) % modu
688 
689             #guarda todas las combinaciones que dan 0
690             #(dependencia linear) mas de una combinacion
691             if(multiplied == 0):
692                 listaCoeff.append(vec)
693 
694         #eliminar elemento cero de la lista (0,0,0,...,0_mod)
695         del listaCoeff[0]
696 
697         #escoger el elemento monico (primer elemento de la
698         #lista sea 1)
699         rows = MT.nrows()
700         l = rows - 1
701         k = None
702         if (listaCoeff):
703             i = 1
704             ran = len(listaCoeff)
705             k = listaCoeff[0]
706             for i in range(0, ran):
707                 element = listaCoeff[i]
708                 if ((element[l] < k[l]) and (element[l] != 0)):
709                     k = listaCoeff[i]
710                     #k son loscoeficientes del polinomio valido
```

```
711
712         #GUARDAR POLINOMIO VALIDO
713
714         #si k existe, guardar a variable global PV y actualizar
715         #los trackers
716         if(k != None):
717
718             #save Valid Polynomial
719             #show("coeficientes = ", k)
720             #show("variables =", MultRowTracker)
721             PolinomioValido(k, MultRowTracker)
722             #coeficientes y pares ordenados (x,y) monomio
723
724             #elimina ultima fila de la tabla
725             Trows = Table.nrows() - 1
726             Table = Table.delete_rows([Trows])
727
728             #elimina multiplos (del ultimo elemento de rowTracker)
729             #de la lista de mult
730             lastMultPlace = (len(MultRowTracker)) - 1
731             parOrdenado = MultRowTracker[lastMultPlace]
732             MultList = delete_multiplo_off_list(MultList,
    parOrdenado)
733
734             #elimiina ultimo elemento de rowTracker
735             del MultRowTracker[-1]
736
737             show("Valid Polynomials = ", PV)
738
739     #***termina busqueda de polinomios validos en el arreglo
740     #***multidimensional
741
742     #escribir en el archivo el final de la sesion y cerrar archivo
743     f.write("          Val(A)\nEnd of session. Time: ")
744     f.write(strftime("%Y-%m-%d %H:%M:%S\n\n\n", gmtime()))
745     f.close()
746     return PV
```

5.2 Archivo text

```
Start of session. Time: 2016-04-28 19:55:16
 sequence s: [0, 0, 0, 1, 0, 1, 1]
 shifts   t: [1, 3, 2, 6, 4, 5]

  y^4 + y^2 + y + 1 element of Val(s)

         A =
 [  1  1  0  0  0  0  ]
 [  0  0  1  1  0  0  ]
 [  1  0  0  1  0  1  ]
 [  0  0  0  0  1  1  ]
 [  0  1  0  1  1  0  ]
 [  0  1  1  0  0  1  ]
 [  1  0  1  0  1  0  ]

 { y^4 + y^2 + y + 1
 y^3 x + y^2 x + y^3 + y^2 + x + 1
 x^6 + 1
 } elements of Val(A)
End of session. Time: 2016-04-28 20:02:41

Start of session. Time: 2016-04-28 23:09:21
 sequence s: [0, 0, 1, 1, 0]
 shifts   t: [1, 3, 4, 2]

 y^4 + y^3 + y^2 + y + 1  element of Val(s)

       A =
 [  1  0  0  1  ]
 [  1  0  0  0  ]
 [  0  0  1  0  ]
 [  0  1  1  0  ]
 [  0  1  0  1  ]

 { y^4 + y^3 + y^2 + y + 1
 x^4 + 1
 } elements of Val(A)
End of session. Time: 2016-04-28 23:16:13

Start of session. Time: 2016-04-29 17:58:54
 sequence s: [0, 1, 1, 0, 1, 0, 0]
 shifts   t: [1, 3, 2, 6, 4, 5]

  y^4 + y^3 + y^2 + 1  element of Val(s)

         A =
 [  0  0  1  0  1  1  ]
 [  1  1  0  0  1  0  ]
 [  0  1  1  0  0  0  ]
 [  1  0  1  1  0  0  ]
 [  1  0  0  0  0  1  ]
 [  0  0  0  1  1  0  ]
 [  0  1  0  1  0  1  ]

 { y^4 + y^3 + y^2 + 1
```

```
 y^3 x + y^3 + y x + x + y + 1
 x^6 + 1
 } elements of Val(A)
End of session. Time: 2016-04-29 18:06:19
```

References

[1] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*. Undergraduate Texts in Mathematics. Springer, New York, second edition, 1997. An introduction to computational algebraic geometry and commutative algebra.

[2] I. Rubio, R. Arce-Nazario, F. Castro, O. Moreno, J. Ortíz. Construction and analysis of multidimensional periodic arrays, draft.

[3] C. Heegard, I. Rubio, and M. Sweedler. Finding a Groebner basis for the ideal of recurrence relations on m-dimensional periodic arrays. In *Comtemporary Developments in Finite Fields and Their Applications (accepted)*, 2015.

[4] O. Moreno and A. Tirkel. Multi-Dimensional Arrays for Watermarking. In *Information Theory Proceedings (ISIT), 2011 IEEE International symposium on*, pages 2691-2695. IEEE, 2011.

[5] O. Moreno and A. Tirkel. New optimal sequences for wireless communications. In *Sequences and their applications–SETA 2012*, volume 7280 of *Lecture Notes in Comput. Sci.*, pages 212-223. Springer, Heidelberg, 2012.

[6] C. Ding, T. Helleseth, W. Shan (1998). On the linear complexity of Legendre Sequences, IEEE Transactions on Information Theory, Vol. 44, No. 3, (pp. 1276-1278).

[7] D. Gómez, T. Høholdt, O. Moreno, I. Rubio (2015). Linear Complexity for multidimensional arrays- a numerical invariant. Proceedings of the IEEE International Symposium on Information Theory (ISIT 2015), (pp. 2697-2701).

[8] Mullen, G. L., Panario, D. (2013). Handbook of finite fields. London, NY: CRC Press.