Linear algebra for computing a recurrence relation of Multidimensional Sequences.

Maizia Mohamed

30/05/2021

Contents

1 Introduction

One of the fundamental problems in computer science is the estimation of the linear complexity of an infinite sequence which is the shortest length of a recurrence relation with constant coefficients that is satisfied by said sequence.This problem's solution has many application in Computer algebra[JCF11], Combinatorics[MBM00] and in error correcting codes[Sak90].
In the one dimensional case the main algorithm used to solve this problem is the Berlekamp-Massey algorithm (BM, Berlekamp (1968) [E.B68];

Massey (1969)[Mas69]). An extension of it is the Berlekamp-Massey-Sakata (BMS ,Sakata 1989,1990[Sak90]) algorithm and another algorithm called Scalar-FGLM[JB15] are the main ones used to solve this problem in the n-dimensional case.
The goal of this project is to familiarize ourselves with n-dimensional tables and to understand how to find the recurrence relation of said tables using linear algebra, then to implement an algorithm capable of that task in C.
The general organization of the project is as follows : we first study the one dimensional case and in particular the Berlekamp-Massey algorithm and provide and implementation of it in Sage[1]. We move on then to study the two-dimensional case as a simplification of the n-dimensional case. In that context we study the modeling through linear systems used to compute the recurrence relation . We study the properties of the different monomial orderings and the linear systems constructed from them .Finally we create an algorithm similar to scalar-FGLM algorithm to solve the problem. Due to lack of time we could not reach that last step.

2 One-dimensional case: Naive algorithm

We start first by defining a recurrence relation. A recurrence relation is an equation that recursively defines a sequence. Once given the initial terms the rest of them are defined as a function of the previous terms. In this project we are only interested in linear recurrence relations. Which mean that our relations are all of the following form in the one-dimensional case :

$$U_{i+d} = \gamma_{i+d-1}U_{i+d-1} + \gamma_{i+d-2}U_{i+d-2}... + \gamma_i U_i$$

Were we have $i, d, \gamma_i \in \mathbb{N}$.
In this section we first started by creating a naive algorithm to solve the problem. Its principal is as follows : we create a Hankel matrix A using the successive terms of the sequence given up to a specified $d \in N$ which is the length of the recurrence relation we are looking for :

$$A = \begin{pmatrix} U_0 & U_1 & ... & U_d \\ U_1 & U_2 & ... & U_{d+1} \\ .... & & & \\ U_{D-d} & U_{D-d+1} & ... & U_D \end{pmatrix}$$

This matrix will insure that all terms are present in order, so by computing a solution we can find any relation between any d successive terms of the sequence. All we need to do next is to solve $Ax = 0$ to get a vector representing the recurrence relation then check the relation for the rest of the terms given before returning the result, any x we find can be transformed so

[1] All source codes : "https://github.com/maikio7/Psfon"

that the coefficient for U_d is one. however since d is not necessarily known it would require trying every possible value until we have a relation that passes the check. This costly algorithm would require computing the solution for linear system n times in the worst case making its complexity $O(n^4)$ where n is the number of terms received by the algorithm. This algorithm was coded in sage.

Algorithm 1: Naive algorithm

```
Data: T : array of the first D terms of the sequence
Result: X : array representing the smallest recurrence relation
        satisfied by the terms given.
d = 2;
c = true;
while c do
    i = 0;
    while i ≤ D − d do
        j = 0;
        while j ≤ d do
            A[i][j] = T[i+j];
            j ++;
        i++;
    X = Null(A); * This operation both find the relation and checks
     its validity for the rest of the terms given *
    if X is not empty then
        c = false;
return X;
```

We take the famous Fibonacci sequence as an example were every term is the sum of the previous two terms with $U_0 = 0$ and $U_1 = 1$, here are the first few terms $(D = 5) : U_0 = 0 ; U_1 = 1 ; U_2 = 1 \; U_3 = 2 \; U_4 = 3 ; U_5 = 5$¿ Using this naive algorithm we first build the following matrix with $d = 2$:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 2 \\ 2 & 3 \\ 3 & 5 \end{pmatrix}$$

When we try calculating null(A) the algorithm returns no solutions . So the next step is $d = 3$:

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 5 \end{pmatrix}$$

This time the algorithm returns the vector :

$$\begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}$$

which is interpreted as $-U_i - U_{i+1} + U_{i+2} = 0$ giving us our recurrence relation $U_{i+2} = U_{i+1} + U_i$

3 One-dimensional case: Berlekamp-Massey algorithm

The most widely used algorithm for computing linear recurrence relation for one dimensional sequences is the Berlekamp-Massey algorithm. The algorithm is based on the following observation : we can extend the previous linear system by extending A and the the null vector as follows , and keep the same x as a solution:

$$\begin{pmatrix} U_0 & U_1 & ... & U_d \\ U_1 & U_2 & ... & U_{d+1} \\ .... & & & \\ U_{D-d} & U_{D-d+1} & ... & U_D \\ U_{D-d+1} & ... & U_D & 0 \\ ... & & & \\ U_D & 0 & ... & 0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ ... \\ q_{d-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ ... \\ U_{d-1} \\ U_{d-2} \\ ... \\ U_0 \end{pmatrix}$$

The product Ax is equivalent to the product of the following polynomials $P = \sum_{i=0}^{D} U_i x^{D-i}$ and $V = x^d + \sum_{i=0}^{d-1} q_i x^i \bmod x^{D+1}$ the vector C corresponds to the polynomial $R = U_{d-1}x^{d-1} + ...U_0 = P \bmod x^{D+1}$. So there is U and V such that :

$$x^{D+1}U + PV = R$$

with this we can find V and U by using the extended Euclid algorithm stopped at the point where we have deg $V >$ deg R. the resulting V is the polynomial representing the vector x and so it also represents our recurrence relation.

The complexity of the Berlekamp-Massey algorithm is the same as that of the extended Euclidean which is $O(D^2)$ for the naive version and $O(M(D)logD)$ in the fast version $M(D)$ being the complexity of multiplying two polynomials of degree D).

We made an implementation using the naive version of extended Euclidean in sage.

Algorithm 2: Berlekamp-Massey

Data: P: polynomial representing an array of terms of a sequence up to the D'th term

Result: V : polynomial representing the recurrence relation satisfied by the terms

```
R0 = x^(D+1);
R1 = P;
V0 = 0 ;
V1 = 1 ;
U0 = 1;
U1 = 0 ;
while degV1 ≤ degR1 do
    Q = R0 / R1 ;
    X = R0 mod R1 ;
    R0 = R1 ;
    R1 = X ;
    UX = U0 - U1*Q ;
    VX = V0 - V1*Q ;
    U0 = U1 ;
    V0 = V1 ;
    U1 = UX ;
    V1 = VX ;
return V1;
```

Using the same Fibonacci example as in the previous section we first build the polynomial P with $D = 5$ we get : $P = (0x^5)+x^4+x^3+2x^2+3x+5$ and we initialize $R0 = x^6$. by using the algorithm above on these two polynomials we get the result $V1 = -x^2 + x + 1$ which we interpret again as $U_{i+2} = U_{i+1} + U_i$.

4 Two-dimensional case

Since one of the goals of this project is to be familiar with two-dimensional sequences we first starting by studying a simple sequence defined by the following relations $\forall (i, j) \in \mathbb{N}^2$:

$$U_{(i,j+2)} = U_{(i,j)}$$

$$U_{(i+2,j)} = U_{(i,j)}$$

$$U_{(i+1,j+1)} = U_{(i,j)}$$

since we had 3 recurrence relations we first tried to define 3 initial terms like this :

$$U_{(0,0)} = a$$

$$U_{(0,1)} = b$$

$$U_{(1,0)} = c$$

however we quickly realized that $U_{(1,0)}$ and $U_{(0,1)}$ had to be equal because of the following : by replacing (i, j) with $(0, 1)$ in the 2nd equation we get $U_{(2,1)} = U_{(0,1)}$. And by replacing it with $(1, 0)$ in the 3rd equation we get : $U_{(2,1)} = U_{(1,0)}$. So in conclusion we get $U_{(0,1)} = U_{(1,0)}$.The table of terms of this sequence looks like this :

$$\begin{pmatrix} a & b & a & ... \\ b & a & b & ... \\ ... & & & \end{pmatrix}$$

which means that despite having 3 recurrence relations in its definition this sequence only needed 2 initial terms to be defined, so it logically follows that it could be defined by using only 2 relations instead of 3. Following the same reasoning as earlier : by replacing (i, j) in the 2nd equation with $(i, j+1)$ and with $(i + 1, j)$ in the 3rd we get $U_{(i+2,j+1)=U_{(i,j+1)}}$ and $U_{(i+2,j+1)} = U_{(i+1,j)}$. As a result we get :

$$\forall (i, j) \in \mathbb{N}^2, U_{(i,j+1)} = U_{(i+1,j)}$$

This 4th equation coupled with the first one are enough to define this sequence without a need for the 3rd and 2nd equations.
As a conclusion we find that it is possible to optimize the number of initial terms needed beyond what might seem possible at a first glance. To achieve this we will need a way to automatically build linear systems capable upon solving them of finding the optimal recurrence relations that define a given sequence. For this we will need to have a way of ordering all the terms of the sequence even if they are of different dimensions.

5 Monomial orderings:

Monomial orderings are essential to various algorithms and operations performed on polynomials such as division or Gaussian reduction, and in our case it helps us systematically build linear systems using the terms of a given n-dimensional sequence. In the one-dimensional case it is intuitive that monomials are ordered as : $1 < x < x^2 < x^3$... when dividing two polynomials for example. however when having to deal with multiple variables it becomes unclear how to order them. Formally a monomial ordering ">" needs to have the following properties :
1) ">" is a total ordering on $\mathbb{Z}^n$ meaning every two monomials are comparable.
2) if $\alpha > \beta$ then $\forall \gamma$ we have $\alpha + \gamma > \beta + \gamma$

3) ">" is a well-ordering on $\mathbb{N}^n$. meaning that every non empty subset of $\mathbb{N}^n$ has a smallest element under ">".
For our purposes 3 monomial orderings are important : the Lexicographical order (lex), the Graded lex order(grlex), and the Graded reverse lex order (gevlex).

5.1 Lexicographical order

We first start by defining the lex order:
let $\alpha = (\alpha_1, \alpha_2, ...)$ and $\beta = (\beta_1, \beta_2, ...)$ be in $\mathbb{N}^n$. We say that $\alpha >_{lex} \beta$ if the leftmost non zero of $\alpha - \beta$ is positive. In practice this order could be said to be similar to the order of words in a dictionary . Supposing we have 3 variables x , y, z in order, when comparing two monomials lex checks the greater power of x among the two and declares that monomial bigger. If equal it then does the same for y then for z. examples : we have $x^5y >_{lex} x^4y^7z^9$ and we have $x^2yz >_{lex} x^2z^3$.
As we can see the lex order does not take into account the total degree of the monomials being compared . For some purposes it might be needed to take that into account which is why grevlex was invented.

5.2 Graded lexicographical order

the definition of grlex is as follows : let α and β be in $\mathbb{N}^n$, we say that $\alpha >_{grlex} \beta$ if :
$|\alpha| = \sum_{i=0}^{n} \alpha_i > |\beta| = \sum_{i=0}^{n}$ or $|\alpha| = |\beta|$ and $\alpha >_{lex} \beta$
So grlex simply compares monomials based on their total degree and breaks ties by using lex. examples : $x^3yz >_{grlex} x^2y^2z$ and $x^3yz <_{grlex} x^2y^2z^2$

5.3 Graded reverse lexicographical order

grevlex is defined as follows : let α and β be in $\mathbb{N}^n$, we say that $\alpha >_{grevlex} \beta$ if :
$|\alpha| = \sum_{i=0}^{n} \alpha_i > |\beta| = \sum_{i=0}^{n}$ or $|\alpha| = |\beta|$ and the rightmost non zero entry of $\alpha - \beta$ is negative.
In practice grevlex works in a similar manner to grlex except that it focuses on reducing the weaker variables instead of maximizing the stronger ones when two monomials have the same total degree. This order is more efficient then lex in some operations.

5.4 Generalities

Some additional definitions : let " > " be some monomial order and $P = \sum_\alpha a_\alpha x^\alpha$ be a non zero plynomial. we have :
the multidegree of $P = max(\alpha \in \mathbb{Z}^n)$ max being taken with respect to " > "
the leading coefficient of P, $LC(P) = a_{multidree(p)}$,

the leading monomial of P, $LM(P) = x^{multidgree(p)}$,
the leading term of P, $LT(P) = LM(P) \times LC(P)$.

In order to familiarize ourselves with the monomial orders above we had an exercise where we proved the following statements:
- grlex and grevlex are equivalant in the two-dimensional case and different for $n \leq 3$
- $1 < x < x^2 < x^3$... in the only monomial order on $K[x]$.
- let $P \in K[x_1, x_2, ..., x_n]$ and m a monomial. we have $LT(m.P) = m.LT(P)$
- let $P, Q \in K[x_1, x_2, ..., x_n]$. we have $LT(P.Q) = LT(P).LT(Q)$
- let $P_i, Q_i \in K[x_1, x_2, ..., x_n$. we have $\sum_{i=1}^{s} P_i.Q_i$ is not necessarily equal to $LM(P_i).LM(Q_i)$

6 two-dimensional case (continuation)

After familiarizing ourselves with monomial orders we can now systematically build linear systems from the terms of the sequence given , which we can then solve to get the recurrence relations we are looking for. For this section we studied the following sequence :

$$U_{i,j} = (1 + (-1)^i)2^{i+j} + (-1)^i 3^{i+j} + (-1)^i 5^{i+j}.$$

here is a 4x4 matrix of the first few terms:

$$\begin{pmatrix} 4 & 12 & 42 & 162 \\ -8 & -34 & -152 & -702 \\ 42 & 168 & 738 & 3432 \\ -152 & -702 & -3368 & -16354 \end{pmatrix}$$

As mentioned in the previous section the grevlex and grlex are equivalent in the two-dimensional case. So in this section we will build two linear systems using grevlex and lex orders respectively . The method used to build the linear systems is as follows : we first determine the size of the matrix, let's call it d .We then order the first d terms of the sequence using the monomial order chosen. We associate each row i of the matrix with the term i of the sequence following the monomial order. And we do the same things for the columns . Now each element of the matrix is associated with two terms of the sequence, we denote them $U_{(a,b)}$ and $U_{(c,d)}$. This term of the matrix is set equal to $U_{(a+c,b+d)}$.

6.1 Linear system with Grevlex

Following the grevlex ordering, and choosing $d = 4$ as an example , we get the following matrix :

$$\begin{pmatrix} U_{(0,0)} & U_{(0,1)} & U_{(0,2)} & U_{(1,1)} \\ U_{(0,1)} & U_{(0,2)} & U_{(0,3)} & U_{(1,2)} \\ U_{(0,2)} & U_{(0,3)} & U_{(0,4)} & U_{(1,3)} \\ U_{(1,1)} & U_{(1,2)} & U_{(1,3)} & U_{(2,2)} \end{pmatrix}$$

We notice that the matrices generated from the grevlex are symmetrical which could help during calculations. For the terms of our sequence described above we used a python script to compute a 10x10 matrix using the method above with the grevlex order. The linear system that we need to solve is $Ax = 0$ meaning we need to compute the kernal of our matrix. which we did by using the null(A,"r") command in Matlab.

Algorithm 3: grevlex

Data: U:matrix of the terms of the two-dimensional sequence,d:size of linear system

Result: X:matrix containing column vector representing the recurrence relations

```
grevlex = [(0,0)(0,1)(1,0)(0,2)(1,1)(2,0)(0,3)(1,2)(2,1)(3,0)];
i = 0;
while i < d do
    j = 0;
    while j < d do
        (m,n) = grevlex[i] + grevlex[j];
        A[i][j] = U[m][n];
        j = j +1;
    i = i +1;
X = null(A);
return X;
```

By applying this method to our relation defined above we got the following column vectors as a result :

$$\begin{pmatrix} 0 & 0 & -30 & 30 & 30 \\ -2 & 0 & 31 & -35 & -35 \\ -2 & 0 & 0 & -4 & -4 \\ 1 & -1 & -10 & 10 & 10 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

which we then interpret to be the following recurrence relations :

$$U_{(i+1,j+1)} = 2(U_{(i,j+1)} + U_{(i+1,j)}) - U_{(i,j+2)}$$

$$U_{(i,j+2)} = U_{(i+2,j)}$$

$$U_{(i,j+3)} = 30U_{(i,j)} + 10U_{(i,j+2)} - 31U_{(i,j+1)}$$

$$U_{(i+1,j+2)} = -30U_{(i,j)} - 10U_{(i,j+2)} + 35U_{(i,j+1)} + 4U_{(i+1,j)}$$

$$U_{(i+3,j)} = -30U_{(i,j)} - 10U_{(i,j+2)} + 35U_{(i,j+1)} + 4U_{(i+1,j)}$$

We can see that the 5th equation can be deduced using the 2nd and 4th equation which makes it redundant . And by replacing (i, j) in the first relation by $(i, j + 1)$ and by using the 3rd relation we can deduce the 4th one as well which also makes it redundant . So we can conclude that the first 3 relations above are the minimum required to define our sequence given.
Going back to our sequence from section 4. Using this method we can build the linear system that allows us to systematically find the relation $U_{i+1,j} = U_{i,j+1}$ that we discovered in that section. We build the following matrix by putting $U_{0,0} = 0$ and $U_{0,1} = 1$ and choosing $d = 4$:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

the result we get this time is :

$$\begin{pmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which is interpreted as the following relations :

$$U_{i,j+1} = U_{i+1,j}$$

$$U_{i,j+2} = U_{i,j}$$

$$U_{i+1,j+1} = U_{i,j}$$

$$U_{i+2,j} = U_{i,j}$$

So as we can see this method allowed us to find all the initial relations of the sequence but also the relation that we had discovered.

6.2 Linear system using Lex

By using the same method used in grevlex we can build a linear system with lex. However in the lex ordering there is an infinite amount of terms between x and x^2 for example, which makes it impossible to reach all the terms of the sequence given. Because of this we need to define a limit on the total degree of monomials . In our case we chose to put it at 2. We use the same algorithm as the previous section except replacing the array grevlex by the array lex = [(0,0)(0,1)(0,2)(1,0)(1,1)(1,2)(2,0)(2,1)(2,2)(3,0)]
hHere is a 9x9 matrix of the terms of a sequence build using the lex ordering :

$$\begin{pmatrix} U_{(0,0)} & U_{(0,1)} & U_{(0,2)} & U_{(1,0)} & U_{(1,1)} & U_{(1,2)} & U_{(2,0)} & U_{(2,1)} & U_{(2,2)} \\ U_{(0,1)} & U_{(0,2)} & U_{(0,3)} & U_{(1,1)} & U_{(1,2)} & U_{(1,3)} & U_{(2,1)} & U_{(2,2)} & U_{(2,3)} \\ U_{(0,2)} & U_{(0,3)} & U_{(0,4)} & U_{(1,2)} & U_{(1,3)} & U_{(1,4)} & U_{(2,2)} & U_{(2,3)} & U_{(2,4)} \\ U_{(1,0)} & U_{(1,1)} & U_{(1,2)} & U_{(2,0)} & U_{(2,1)} & U_{(2,2)} & U_{(3,0)} & U_{(3,1)} & U_{(3,2)} \\ U_{(1,1)} & U_{(1,2)} & U_{(1,3)} & U_{(2,1)} & U_{(2,2)} & U_{(2,3)} & U_{(3,1)} & U_{(3,2)} & U_{(3,3)} \\ U_{(1,2)} & U_{(1,3)} & U_{(1,4)} & U_{(2,2)} & U_{(2,3)} & U_{(2,4)} & U_{(3,2)} & U_{(3,3)} & U_{(3,4)} \\ U_{(2,0)} & U_{(2,1)} & U_{(2,2)} & U_{(3,0)} & U_{(3,1)} & U_{(3,2)} & U_{(4,0)} & U_{(4,1)} & U_{(4,2)} \\ U_{(2,1)} & U_{(2,2)} & U_{(2,3)} & U_{(3,1)} & U_{(3,2)} & U_{(3,3)} & U_{(4,1)} & U_{(4,2)} & U_{(4,3)} \\ U_{(2,2)} & U_{(2,3)} & U_{(2,4)} & U_{(3,2)} & U_{(3,3)} & U_{(3,4)} & U_{(4,2)} & U_{(4,3)} & U_{(4,4)} \end{pmatrix}$$

As we can observe this matrix has a special structure named multi-Hankel . It contains smaller 3x3 matrices each one being a Hankel matrix, and these matrices themselves follow a Hankel structure inside the main matrix.

7 Conclusion

In Summary, during this project we studied the problem of computing recurrence relations starting with the simpler one-dimensional case were we tried a naive algorithm then used the Berlekamp-Massey algorithm for better performance. Studying the two-dimensional case we determined that it was needed for us to have a way to systematically build linear systems to find the recurrence relations. After studying the monomial orders we were able to build said linear systems and got structured matrices that help speed up calculations. In particular our goal was to use the Multi-Hankel structure of the matrix generated using the Lexicographic order along with fast linear algebra methods to compute a solution to this problem efficiently but due to lack of time this part was not achieved.

References

[E.B68] E.Berlekamp. Nonbinary bch decoding. *IEEE Transactions on Information Theory*, 14(2):242–242, march 1968.

[JB15] Jean-Charles Faugere Jeremy Berthomieu, Brice Boyer. Linear algebra for computing gröbner bases of linear recursive multidimensional sequences. *ISSAC*, 40:61–68, 2015.

[JCF11] Chenqi Mou Jean-Charles Faugere. Fast algorithm for change of ordering of zero-dimensional gröbner bases with sparse multiplication matrices. *ISSAC*, pages •115,122, June 2011.

[Mas69] J. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15(1):122 – 127, Jan 1969.

[MBM00] Marko Petkovsekb Mireille Bousquet-Meloua. Linear recurrences with constant coeficients:the multivariate case. *Discrete Mathmatics*, 225:1–3, April 2000.

[Sak90] Shojirou Sakata. Extension of the berlekamp-massey algorithm to n dimensions. *Information and computation*, pages 207–239, 1990.