EML: Linear Algebra over the Integers

Joris LIMONIER & Dany ALVES MARQUES

Supervised by : Gabor WIESE & Luca NOTARNICOLA

March-June 2018

Preface

This article is the result of a three-months project about linear algebra over the integers. The programming part was completed using SageMath and its matrix related features.
This topic appeared particularly appealing as it seemed to allow us to find some nice results and tools while being only in first year. We thought it could bring us some interesting knowledge and useful skills that we would be able to reuse during the rest of our degree. We must say we were not disappointed at all so we will try to properly share our enjoyment while presenting our work in this article.

We wish to thank Prof. Dr. Gabor Wiese and Luca Notarnicola for their guidance throughout the making of this article.

Contents

1 Introduction

Definition 1.1. ***Hermite Normal Form (HNF)*** *We will say that a $m \times n$ matrix $M = (m_{i,j})$ with integer coefficients is in HNF if there exists $r \leq n$ and a strictly increasing map f from $[r+1, n]$ to $[1, m]$ satisfying the following properties :*

(1) For $r + 1 \leq j \leq n$, $m_{i,j} = 0$ if $i > f(j)$ and $0 \leq m_{f(k),j} < m_{f(k),k}$ if $k < j$.

(2) The first r columns of M are equal to 0.

Remark. *In the important special case where $m = n$, and $f(k) = k$($\iff$ $det(M) \neq 0$), M is in HNF if it satisfies the following conditions :*

(1) M is an upper triangular matrix, i.e. $m_{i,j} = 0$ if $i > j$.

(2) For every i, we have $m_{i,i} > 0$.

(3) For every $j > i$, we have $0 \leq m_{i,j} < m_{i,i}$.

Theorem 1.2. *Let $A \in \mathcal{M}_{m,n}(\mathbb{Z})$. Then there exists a unique $m \times n$ matrix $B = (b_{i,j})$ in HNF of the form $B = AU$ with $U \in GL_n(\mathbb{Z})$ where $GL_n(\mathbb{Z})$ is the group of matrices with integer coefficients which are invertible, i.e. whose determinant is equal to ± 1.*

Proof. The proof is given by the algorithm in section 2.1 on page 3.

Remark. *The reader should take note that some other litterature may give a different definition for the HNF form of a matrix. This explains the possible difference between results obtained on HNF online calculators and the ones obtained in this paper for instance. For the entire article, a matrix will be in HNF if and only if it is in accordance with the definition 1.1.*

Here are a few example to set down what being in HNF means. We will see a couple cases with different numbers of lines and columns:

Example 1.3.

a) *Case of a* 2×2 *matrix :* $M_1 = \begin{pmatrix} 5 & 7 \\ 2 & 3 \end{pmatrix}$
M_1 is not in HNF because it is a square matrix which is not upper triangular.

b) *Case of a* 2×5 *matrix :* $M_2 = \begin{pmatrix} 0 & 0 & 2 & 6 & 13 \\ 0 & 0 & 0 & 7 & 1 \end{pmatrix}$ *is not in HNF because the first r columns from defninition 1.1 are not equal to 0 (i.e. $M_{1,3} = 2 \neq 0$).*

c) *Case of a* 4×4 *matrix. :* $M_3 = \begin{pmatrix} 4 & 8 & 5 & 11 \\ 7 & 2 & 6 & 1 \\ 1 & 17 & 7 & 12 \\ 9 & 3 & 5 & 8 \end{pmatrix}$ *is not in HNF because it is a square matrix which is not upper triangular.*

d) *Case of a* 3×5 *matrix. :* $M_4 = \begin{pmatrix} 0 & 0 & 51 & 50 & 11 \\ 0 & 0 & 0 & 28 & 2 \\ 0 & 0 & 0 & 0 & 13 \end{pmatrix}$ *is in HNF because it fits the criteria from definition 1.1 (where $r = 2$).*

2 A systematic HNF computation

In the first subsection, we will present the HNF algorithm in a spoken-like way. By doing so, we intend to making it easier for the reader to understand the procedure.
Then, in the following subsection we will present and explain the actual SageMath code and the functions used.

2.1 Algorithm

Algorithm 2.1. *Given an $m \times n$ A with integer coefficients $(a_{i,j})$, this algorithm finds the HNF W of A. Let us call $(w_{i,j})$ for the coefficients of W, A_i (resp. W_i) for the i^{th} column of A (resp. W)*

Step 1. [Initialize] Set $i \leftarrow m$, $k \leftarrow n$, $\begin{cases} l = 1 & if \quad m \leq n \\ l = m - n + 1 & if \quad m > n \end{cases}$

Step 2. [Row finished ?] If all the $a_{i,j}$ with $j < k$ are zero, then if $a_{i,k} < 0$, replace column A_k with $-A_k$ and go to step 5.

Step 3. [Choose non-zero entry] Pick among the non-zero $a_{i,j}$ for $j \leq k$ one with the smallest absolute value, say a_{i,j_0}. Then if $j_0 < k$, exchange column A_k with column A_{j_0}. In addition, if $a_{i,k} < 0$, replace column A_k by $-A_k$. Set $b \leftarrow a_{i,k}$.

Step 4. [Reduce] For $j = 1, \ldots, k-1$ do the following : set $q \leftarrow \lfloor a_{i,j}/b \rceil$ and $A_j \leftarrow A_j - qA_k$. Then go to step 2.

Step 5. [Final reductions] Set $b \leftarrow a_{i,k}$. If $b = 0$, set $k \leftarrow k + 1$ and go to step 6. Otherwise for $j > k$ do the following: set $q \leftarrow \lfloor a_{i,j}/b \rfloor$, and $A_j \leftarrow A_j - qA_k$.

Step 6. [Finished?]If $i = l$, then for $j = 1, \ldots, n - k + 1$ set $Wj \leftarrow A_{j+k-1}$ and terminate the algotithm. Otherwise, set $i \leftarrow i - 1$, $k \leftarrow k - i$ and go to step 2.

This algorithm terminates since the $\mid a_{i,k} \mid$ are streactly decreasing every time we return from **Step 2** to **Step 4**. It is clear that upon termination W will be in accordance to the HNF definition given in definition 1.1. Moreover, as we only used elementary columns operations on A, W is the HNF of A. □

2.2 The code

The code which will be given in this subsection will be added as figures and the comments/explanations will be within the figures.

⚠ We would like to bring to the attention of the reader that Sagemath starts counting rows and columns from 0. It is crucial to keep this point in mind as it means a matrix of size $m \times n$, according to SageMath, has rows numbered from 0 to m-1 and columns numbered from 0 to n-1.

One symbol "#" will be added before the first line of a comment (this is the normal way to add a comment on SageMath). But if a comment goes through several lines, we will use the following convention: two "#" for the second line, three "#" for the third line and so on. Here is an example to clarify the convention :

#comment number 1 (1st line)
#comment number 2 (1st line)
#comment number 2 (2nd line)
#comment number 3 (1st line)

```
#Let A be a mxn matrix
##this is a "random" matrix to set ideas down.

A=matrix([[21, 2, 3, 4, 5, 6],[7, 8, 9, 10, 11, 12],[13, 14, 15, 16, 17, 18]])
print A
i=A.nrows()-1 #set i as last row of matrix
k=A.ncols()-1 #set k as last column of matrix

#Initialise l:
if A.nrows()<=A.ncols():
    l=0
else:
    l=i-k
j=0

#Here starts the HNF Process:
#Step 1
while i!=l:

    #We define a row-matrix r such that it has all the elements A[i][j] of the i-th row, with j<k
    r=A.matrix_from_rows_and_columns([i],range(0,k))
    print"This is r"
    print r

    #Step 2
    while r.is_zero()!=True:
        #Define a row-matrix y that has the absolute value of all the
        ##elements A[i][e] of the i-th row, with: e<=k & A[i][e] !=0
        e=0
        y=[]
        while e<=k:
            if A[i][e]!=0:
                y.append(abs(A[i][e]))
            e=e+1
        print "This is y"
        print y

        #Step 3
        #Find minimum:
        z=0
        if len(y)>=2:
            jo=min(y[z],y[z+1]) #if len(y)=2 we take the minimum of the two elements of y
                                #We still take the minimum of the first two elements if len(y)>2 (see next while loop)
                                #We call the minimum jo
            z=2
            while z<=len(y)-1: #This is a while loop that will find the minimum of y for the case len(y)>2
                jo=min(jo,y[z]) ##as we found the minimum of the first two (called jo)

                                #we overwrite jo by saying that is the minimum of (the minimum of the elements
                                ##before) jo and the z-th element
                z=z+1
        else:
            if len(y)==1: #If y has only one element the minimum is the element itself
                jo=y[0]
        print "this jo "
        print jo
```

Figure 1: The entire HNF code (part 1/4).

```
57
58          #Now we search the column u to which jo belongs (from the 0-th to the (n-1)-th)
59          u=0
60          while abs(A[i][u])!=jo:
61              u=u+1
62          #if u is different from k we exchange column u and column k.
63          if abs(A[i][k])!= jo:
64              if u<k:
65                  A.swap_columns(u,k)
66
67          #column Ak=-Ak if aik is negative
68          if A[i][k] < 0:
69              A.rescale_col(k,-1)
70          print "new A"
71          print A
72
73          #define b=a_{ik} (usually from all the steps before: b is smallest
74          ##positive integer, but b can also be 0 in some cases )
75          b = A[i][k]
76
77
78          #Step 4
79          z=0
80          #Az= Az -q*Ak for all integer  0<=z<k
81          while z <= k-1 :
82              q = round(A[i][z]/b)
83              print "this is q1"
84              print q
85              A.set_column(z,A.column(z)-q*A.column(k))
86              z = z+1
87          print A
88
89          #As this is a loop we need to redefine r such that we can check again the condition
90          r=A.matrix_from_rows_and_columns([i],range(0,k))
91          print "This is r after checking the condition (see comments)"
92          print r
93
94      #If r==0 we change column Ak=-Ak if a_{ik} is negative.
95      if (r.is_zero()==True)or(k==0) :
96          if A[i][k] < 0:
97              A.rescale_col(k,-1)
98          b = A[i][k]
99
100
101         #Step 5:
102         if b==0:
103             k=k+1
104
105         else:
106             #check if there exists at least one column after column k
107             if k < A.ncols()-1 :
108                 w=k+1
109             # Aj<- Aj-q*Ak  for j>k (not the same q as before )
110                 while w<=A.ncols()-1:
111                     q = floor(A[i][w]/b)
112                     A.add_multiple_of_column(w,k,-q)
113                     w=w+1
```

Figure 2: The entire HNF code (part 2/4).

```
    k=k-1   #as the whole algorithm is a while-loop we need to redo this step until l=i
    i=i-1
#This happens when i=l, as i=l is the condition to break the loop
##above, but we still need to check the last step one last Time

#Final Stage
if i==l:
    print "Final Stage"
    #We define a row-matrix r such that its has all the elements A[i][j] of the i-th rows, with j<k
    r=A.matrix_from_rows_and_columns([i],range(0,k))
    print"This is r"
    print r

    #Step 2
    while r.is_zero()!=True:
        #Define a list/row-matrix y that has the absolute value of all the elements
        ##A[i][e] of the i-th row, with: e<=k & A[i][e] !=0
        e=0
        y=[]
        while e<=k:
            if A[i][e]!=0:
                y.append(abs(A[i][e]))
            e=e+1
        print "This is y"
        print y

        #Find minimum(Step3):
        z=0
        if len(y)>=2:
            jo=min(y[z],y[z+1]) #if len(y)=2 we take the minimum of the two elements of y
                                #We still take the minimum of the first two elements if len(y)>2
                                #We call the minimum jo
            z=2
            while z<=len(y)-1: #Here is a loop that will find the minimum of y for the case len(y)>2
                jo=min(jo,y[z]) #as we found the minimum of the first two (called jo)
                                #we overwrite jo by saying that is the minimum of
                                ##(the minimum of the elements before) jo and the z-th element
                z=z+1
        else:
            if len(y)==1: #If y has only one element the minimum is the element itself
                jo=y[0]
        print "this is jo "
        print jo
        #Next step we search the column u which has that element jo (from left to right)
        u=0
        while abs(A[i][u])!=jo:
            u=u+1
        #if u is different from k we exchange column u and column k
        if abs(A[i][k])!= jo:
            if u<k:
                A.swap_columns(u,k)

        #column Ak=-Ak if a_{ik} is negative.
        if A[i][k] < 0:
            A.rescale_col(k,-1)
        print "new A"
        print A
```

Figure 3: The entire HNF code (part 3/4).

```
        #define b=aik (usually from all the steps before: b is smallest
        ##positive integer, but b can also be 0 in some cases )
        b = A[i][k]
        #Step 4
        z=0
        #Az= Az -q*Ak for all integer  0<=z<k
        while z <= k-1 :
            q = round(A[i][z]/b)
            print "this is q1"
            print q
            A.set_column(z,A.column(z)-q*A.column(k))
            z = z+1
        print A

        #As this is a loop we need to redefine r st: we can check again the condition
        r=A.matrix_from_rows_and_columns([i],range(0,k))
        print "This is r after some computations"
        print r

    #If r==0 we change column Ak=-Ak if aik is smaller than 0
    if (r.is_zero()==True)or(k==0) :
        if A[i][k] < 0:
            A.rescale_col(k,-1)
        b = A[i][k] #b=aik
        #Step 5:
        if b==0:
            k=k+1

        else:
            #check if there exists atleast one column after column k
            if k < A.ncols()-1 :
                w=k+1
            # Aj<- Aj-q*Ak  for j>k (not the same q as before )
                while w<=A.ncols()-1: #Colonne pour ligne
                    q = floor(A[i][w]/b)
                    A.add_multiple_of_column(w,k,-q)
                    w=w+1
    #Define W the hermit-normal-form of A that starts from the k-th column as everything before is 0
    W=A.matrix_from_columns(range(k,A.ncols()))
    print "This is W"
    print W
```

Figure 4: The entire HNF code (part 4/4).

2.3 Applications using the HNF

The process computing the HNF of a matrix can be very useful in several domains. In this subsection we will give some of these domains with a brief explanation of the use in each case.

2.3.1 Linear Diophantine equations

The HNF appears to be very useful for solving systems of linear Diophantine equations.

Let A be a matrix and b a vector. We are looking for a vector x such that $Ax = b$. In fact, we may assume that A has full row rank; otherwise, we may remove redundant equations from the system. Yet, we assume all input

data to be rational.

We can find a unimodular matrix U such that $[H \mid 0] = AU$ is a matrix in Hermite normal form. Now, we can apply a standard trick to transform a system of linear equations (and actually, a system of linear inequalities, too) into a more suitable form: Ax = b is equivalent to $(AU)(U^{-1}x) = b$, and therefore, $[H \mid 0]z = b$, where $z = U^{-1}x$ is integral if and only if x is integral. We can observe that the last components of z may take arbitrary values, while feasibility of the system $[H \mid 0]z = b$, and therefore, of Ax = b, depends only on whether the vector $H^{-1}b$ is integral.
From there, we can find ways to set conditions for a system of linear Diophantine equations to have a solution.

We will not go any further into explaining how the HNF is used to solve Linear Diophantine equations as this subsection is just meant to give leads on some applications. Further information can be found on some of the references listed at the end of this article.

2.3.2 case Ax=0

The HNF can also be used to determine the kernel of a matrix. Here is the code to do so :

```
A=matrix([[21, 2, 3],[7, 8, 9],[13, 14, 15]])#Define a Matrix mxn
print A
#Initalise:
i=A.nrows()-1 #i=m
j=A.ncols()-1 #j(=k)=n
k=A.ncols()-1
U=matrix.identity(A.ncols()) #U=In(Identity matrix nxn)
#set l:
if A.nrows()<=A.ncols():
    l=0
else:
    l=i-k
#Ax=0 Process:
while i!=l:
    #We finish when i=l (l depending on matrix A)
    while j!=0:
        #Algorithm until j=0
        j=j-1
        if A[i][j]!=0:
            #We compute xgcd= extended Euclid Algorithm
            a=xgcd(A[i][k],A[i][j])
            d=a[0] #d=gcd(aij,aik)
            u=a[1] #u first integer
            v=a[2] #v second integer
            #Computations on columns of A and U
            #We want to exchange values(The reason we create the temporary vector B)
            B=vector(u*A.column(k)+v*A.column(j)) #B defined such that we have for i-th row the gcd of column j and k of A
            #We want to exchange Values (The reason we create the temporary vector C)

            #Important Step: Inorder to keep the values of aik and aij we first compute this step on U and then on A

            C=vector(u*U.column(k)+v*U.column(j)) #C defined as B in the Beginning but for our matrix U
            U.set_column(j,(A[i][k]/d)*U.column(j)-(A[i][j]/d)*U.column(k)) #Uj=aik/d*Uj-aik/d*Uk
            U.set_column(k,C) #Uk=C
            A.set_column(j,(A[i][k]/d)*A.column(j)-(A[i][j]/d)*A.column(k)) #set Aj <- aik/d *Aj - aij/d*Ak
            A.set_column(k,B) #Ak = B
            print "New A"
            print A
            print "U:"
            print U


    #if aik<0 Ak=-Ak, Uk=-Uk
    if A[i][k] < 0:
            A.rescale_col(k,-1)
            U.rescale_col(k,-1)
    #We define b = aik
    b=A[i][k]
    #check if b=0 or not
    if b==0:
        k=k+1
    else:
        #check if there exists atleast one column after column k
            if k<A.ncols()-1 :
                r=k+1
```

Figure 5: The kernel code (part 1/2).

```
            # Aj<- Aj-q*Ak  for j>k (here j is r)
                while r<=A.ncols()-1: #Colonne pour ligne
                    q = floor(A[i][r]/b)
                    A.add_multiple_of_column(r,k,-q)
                    U.add_multiple_of_column(r,k,-q)
                    r=r+1
    #As this a while loop we define it like this
    i=i-1
    k=k-1
    j=k
#Final Step: i==l

if i==l:
    while j!=0: #Algorithm until j=0
        if A[i][j]==0: #if aij =0 we "move on"
            j=j-1
        else: #otherwhise we also set j=j-1 like before
            j=j-1
            #We compute xgcd= extended Euclid Algorithm
            a=xgcd(A[i][k],A[i][j])
            d=a[0] #d=gcd(aij,aik)
            u=a[1] #u first integer
            v=a[2] #v second integer
            #Computations on columns of A and U
            #We want to exchange values(The reason we create the temporary vector B)
            B=vector(u*A.column(k)+v*A.column(j)) #B defined such that we have for i-th row d (gcd(aik,aij))
            #We want to exchange Values (The reason we create the temporary vector C)
            C=vector(u*U.column(k)+v*U.column(j)) #C defined as B in the Beginning but for our matrix U
            U.set_column(j,(A[i][k]/d)*U.column(j)-(A[i][j]/d)*U.column(k)) #Uj=aik/d*Uj-aik/d*Uk
            U.set_column(k,C) #Uk=C
            A.set_column(j,(A[i][k]/d)*A.column(j)-(A[i][j]/d)*A.column(k)) #set Aj <- aik/d *Aj - aij/d*Ak
            A.set_column(k,B) #Ak = B
            print A
            print "U:"
            print U
    #if aik<0 Ak=-Ak, Uk=-Uk
    if A[i][k] < 0:
            A.rescale_col(k,-1)
            U.rescale_col(k,-1)
    #We define b = aik
    b=A[i][k]
    #check if b=0 or not
    if b==0:
        k=k+1
    else:
        #check if there exists atleast one column after column k
            if k<A.ncols()-1 :
                r=k+1
            # Aj<- Aj-q*Ak  for j>k (here j is r)
                while r<=A.ncols()-1: #Colonne pour ligne
                    q = floor(A[i][r]/b)
                    A.add_multiple_of_column(r,k,-q)
                    U.add_multiple_of_column(r,k,-q)
                    r=r+1
    #We define a basis of the kernel of A with kerA=Uj and j=0,...,k (1,...,k-1)
    kerA=U.matrix_from_columns(range(0,k))
    print "This is the kernel of A"
    print kerA
```

Figure 6: The kernel code (part 2/2).

References

- https://en.wikipedia.org/wiki/Hermite_normal_form
- https://disopt.epfl.ch/webdav/site/disopt/shared/IntPoints2009/lattices.pdf
- https://disopt.epfl.ch/webdav/site/disopt/shared/IntPoints2009/hnf.pdf
- https://ac.els-cdn.com/0024379590902285/1-s2.0-0024379590902285-main.pdf?_tid=b6c4503c-cd0e-45c6-8050-f884bfe51abc&acdnat=1528063809_91dde5a4a0594a39b55b29edfce05eb7
- https://www.uow.edu.au/ thomaspl/pdf/Tourloupis13.pdf

Appendix

This appendix is meant to explain the SageMath built-in functions used in the HNF code (section 2.2) and/or in the code used to find the Kernel of a matrix.

Let A be a matrix of size $(m \times n)$.

A.nrows() : gives the number of rows the matrix A has $(= m)$

A.ncols() : gives the number of columns the matrix A has $(= n)$

A.matrix_from_rows_and_columns($[i_1, i_2, \ldots, i_p], [j_1, j_2, \ldots, j_q]$**)** : creates a submatrix of A created from the rows and columns selected. Where $0 \leq i_a \leq (m-1)$ is the a-th line and $0 \leq j_b \leq (n-1)$ is the b-th column. Lines and Columns can be reordered or repeated.

A.matrix_from _ columns($[j_1, j_2, \ldots, j_q]$**)** : command A.matrix_from_rows_and_columns (see before) where all rows are kept and stay in the same order.

A.swap_columns(u,k) : exchanges the column u and column k with $0 \leq u, k \leq m-1$.

A.rescale_col(k,s) : multiplies the column k $(0 \leq k \leq m-1)$ by s. This command only works if s matches

the ring of A.
Example : ring of $A = \mathbb{ZZ}$ and $s = 0.5 \Rightarrow ERROR$

A.set_column(i,u) : set the i-th column to u with u being a vector of size $(n \times 1)$.

xgcd(i,j) : Computes the Extended Euclidean Algorithm to find coefficients u and v such that: $u \times i + v \times j = gcd(i, j)$ returns $(gcd(i, j), u, v)$. Caution: $\mid u \mid$ and $\mid v \mid$ are not minimal.

A.add_multiple_of_column(r,k,q) : replaces the r-th column by $r + q \times k$ where q belongs to A's ring and k stands for the k-th column $(0 \leq k \leq m - 1)$.

A.is_zero() : returns true if matrix A is the zero matrix.