IRASA International Scientific Conference
SCIENCE, EDUCATION, TECHNOLOGY AND INNOVATION
SETI VI 2024
Book of Proceedings

DOI: `https://doi.org/10.62982/seti06.tami.11

AN ALGORITHM IN THE PYTHON PROGRAMMING LANGUAGE FOR DETERMINING THE JORDAN, SMITH, RATIONAL (FROBENIUS) AND HERMITE NORMAL CANONICAL FORMS

Tatjana Mirković [41]

Abstract

In this paper, the author presents a theory about the Jordanian, Smith rational and Hermite normal canonical forms. Software for determining these canonical forms is provided. Algorithms, written by the author in the Python programming language, are given.

Key words: *Jordan, Smith and rational (Frobenius) and Hermite normal canonical form*

Introduction

The canonical forms of which we speak are generic expressions for the representation of symmetric or skew-symmetric tensors over a finite-dimensional vector space. The theory of canonical forms itself was actively pursued near the end of the nineteenth century through the early part of the twentieth, chiefly by British and German algebraists, and mainly for symmetric tensors.
H.J.S. Smith in 1861 defined the invariant factors of an integer matrix, and obtained the Smith normal form of these matrices. Previously Echelon forms over fields (one-sided reduction) were studied by J.C.F. Gauss. The normal form for similarity classes was described by C. Jordan in his "Traite des substitutions" in 1870.
Four forms studied here are the Jordan canonical form, the Smith normal form, the rational canonical form and Hermite normal form. First, we give the short definitions of these canonical forms:
Definition 1. The Smith normal form of an integer matrix $A \in M_{m\times n}(\mathbb{Z})$ is a factorization $A = UDV$ where:

1. $D \in M_{m\times n}(\mathbb{Z})$ is "diagonal", meaning that $D_{ij} = 0$ whenever $i \neq j$.
2. Each diagonal entry of D divides the next: $D_{ii}|D_{i+1.i+1}$. These diagonal entries are called the elementary divisors of A.
3. $U \in M_{m\times n}(\mathbb{Z})$ and $V \in M_{m\times n}(\mathbb{Z})$ are invertible over $\mathbb{Z}$, or equivalently $detU$ and $detV$ are ± 1.

41 Tatjana Mirković, Academy of Applied Studies Sabac, Sabac, Republic of Serbia, tmirkovic75@gmail.com

Definition 2. Rational canonical (or Frobenius normal) form for matrices Any square matrix M over k is similar to a block diagonal one for which the blocks are companion matrices $C\left(p_i^{n_i}\right)$ of powers of irreducible polynomials p_i over k. Two such block diagonal matrices are similar if and only if one can be obtained from the other by reordering the blocks. Jordan canonical form for matrices

Definition 3. Any square matrix M over a field k containing all of its eigenvalues is similar to a block diagonal matrix with Jordan blocks corresponding to eigenvalues of M. Two such block diagonal matrices are similar if and only if one can be obtained from the other by reordering the blocks.

Definition 4. An $m \times n$ matrix $H = \left(h_{i,j}\right)$ is in Hermite Normal Form if there exists $r \leq m$, and a strictly increasing map $f: [r+1] \to [1, m]$ satisfying the following properties.

1. For $r+1 \leq j \leq m$, $h_{f(j),j} \geq 1$, $h_{ij} = 0$ if $i > f(j)$ and $0 \leq h_{f(k),j} < h_{f(k),k}$ if $k < j$.
2. The last $m - r$ rows of H are equal to 0.

Normal canonical forms

In this chapter, we give the theory from canonical normal forms.

The Jordan Normal (Canonical) Form

Definition 1. Let A be an $m \times m$ matrix and $\lambda \in \mathbb{C}$. Then

1. $m_A(\lambda) = mult_\lambda(ch_A)$, the multiplicity of λ as a root of $ch_A(t)$ is called the algebraic multiplicity of λ in A.
2. $\nu_A(t) = \dim_C E_A(\lambda)$ is called the geometric multiplicity of λ in A.

Here, $E_A(\lambda) = \{\vec{v} \in \mathbb{C}^m : A\vec{v} = \lambda\vec{v}\} = Nullsp(A - \lambda I)$ denotes the λ-eigenspace of A.

Definition 2. Given a scalar $\lambda \in \mathbb{C}$ and an integer $k \geq 1$, the Jordan block of order k corresponding to λ is the matrix $J_k(\lambda)$ of the form:

$$J_k(\lambda) = \begin{bmatrix} \lambda & 0 & \cdots & 0 \\ 0 & \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda \end{bmatrix}$$

Definition 3. If $A \in \mathcal{M}_n(\mathbb{C})$ then there exists an invertible matrix $P \in \mathcal{M}_n(\mathbb{C})$ and Jordan blocks $J_{k_1}(\lambda_1), J_{k_2}(\lambda_2), \ldots, J_{k_r}(\lambda_r)$ such that

$$A = P \begin{bmatrix} J_{k_1}(\lambda_1) & 0 & \cdots & 0 \\ 0 & J_{k_2}(\lambda_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & J_{k_r}(\lambda_r) \end{bmatrix} P^{-1}$$

This block diagonal matrix is called the Jordan canonical form of A. Moreover:

• $\lambda_1, \lambda_2, \ldots, \lambda_s$, are the (distinct) eigenvalues of A;

• The number of Jordan blocks $J_{i_1}, J_{i_2}, \ldots$ with eigenvalue λ_i equals the geometric multiplicity $\nu_A(\lambda_i) = \nu_i$;
• The sum of the sizes k_{ij} of the blocks $J_{i_1}, J_{i_2}, \ldots, J_{i,\nu_i}$ with eigenvalue λ_i equals the algebraic multiplicity $m_i = m_A(\lambda_i)$:

$$k_{i1} + k_{i2} + \ldots + k_{i\nu_1} = m_i.$$

• The J_{ij}'s are uniquely determined by A up to order.

Remark 1. If J and J' are two Jordan matrices which have the same lists of Jordan blocks but in a different order, then J and J' are similar, i.e. there is a matrix P such that $J' = P^{-1}JP$.
We consider: How to find P such that $A = PAP^{-1}$ (for $m = 3$)?

In order to find $P = (\vec{v}_1|\ldots|\vec{v}_n)$ such that $A = PAP^{-1}$ write this equation as $AP = PJ$. By using the identities

$$AP = (A\vec{v}_1|\ldots|A\vec{v}_n),$$

$$P = (a_1|\ldots|a_n)^t = a_1\vec{v}_1 + \ldots + a_n\vec{v}_n,$$

the equation $AP = PJ$ translates into a set of (vector) equations for the $\vec{v}_i$'s which we can solve. The following examples show how this method works.

Remark 2. Suppose J and J' are two Jordan matrices which are similar. Then the lists of Jordan blocks of J and J' are the same (up to order).
Corollary 1. Two $m \times m$ matrices A and B are similar (that is, $B = P^{-1}AP$ for some P) if and only if they have the same Jordan canonical form (up to order) J.

Example 1. If $A = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 2 & 0 \\ 1 & -2 & 3 \end{bmatrix}$, find P such that $P^{-1}AP$ is a Jordan matrix.

Solution.
Step 1. $ch_A(t) = (t-2)^3$. Thus, the only eigenvalue is $\lambda_1 = 2$; its algebraic multiplicity is $m_1 = 3$.

By row reduction we get

$$A - 2I = \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & -2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} 1 & -2 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Thus, the 2-eigenspace is $E_A(2) = \{c_1(2,1,0)^t + c_2(1,0,-1)^t : c_1, c_2 \in \mathbb{C}\}$, and hence $\nu_1 = 2$. Therefore, J has 2 Jordan blocks and

$$J = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Step 2. Find P such that $AP = PJ$. Write $P = (\vec{v}_1|\vec{v}_2|\vec{v}_3)$, where $\vec{v}_1, \vec{v}_2, \vec{v}_3 \in \mathbb{C}^3$. Then we want to choose the $\vec{v}$'s such that $AP = PJ$, i.e. such that $(A\vec{v}_1|A\vec{v}_2|A\vec{v}_3) = (2\vec{v}_1|\vec{v}_1 + 2\vec{v}_2|\vec{v}_3)$. Thus we want:

$$\begin{array}{ll} A\vec{v}_1 = 2\vec{v}_1 & (A-2I)\vec{v}_1 = \vec{0} \\ A\vec{v}_2 = \vec{v}_1 + 2\vec{v}_2, \quad i.e. & (A-2I)\vec{v}_2 = \vec{v}_1. \\ A\vec{v}_3 = 2\vec{v}_3 & (A-2I)\vec{v}_3 = \vec{0} \end{array}$$

In addition, we need to pick the $\vec{v}_i$'s such that $\vec{v}_1$, $\vec{v}_2$ and $\vec{v}_3$ are linearly independent.
Pick $\vec{v}_2 \in E_A^2(2)$, not in $E_A(2)$;
Define $\vec{v}_1 = (A-2I)\vec{v}_2$; Pick $\vec{v}_3 \in E_A(2)$, linearly independent from $\vec{v}_1, \vec{v}_2$.
Now,

$$E_A(2) = \{c_1(2,1,0)^t + c_2(1,0,-1)^t\}$$
$$E_A^2(2) = \mathbb{C}^3.$$

Thus take $\vec{v}_2 = (1,0,0)^t$; Therefore $\vec{v}_1 = (A-2I)\vec{v}_2 = (-1,0,1)^t$; We take $\vec{v}_3 = (2,1,0)^t \in E_A(2)$.
Thus

$$P = \left(\vec{v}_1|\vec{v}_2|\vec{v}_3\right) = \begin{bmatrix} -1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix},$$

and Jordan matrix is

$$J = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

Smith Normal Form

Definition 4. Let $A = (a_{ij})$ be a $k \times n$ matrix with entries in the ring $R = \mathbb{Z}$ of integers. We say that the matrix A is in Smith normal form if

- $a_{ij} = 0$ for $i \neq j$;
- For some m with $0 \leq m \leq k$, $a_{ii} \neq 0$ for $1 \leq m$ and $a_{ii} = 0$ for $i > m$;
- Let $a_i = a_{ii}$ for $1 \leq i \leq m$. Then $a_1|a_2|\ldots|a_m$.

If $m = 0$ in the above definition, then the matrix A in Smith normal form is the zero matrix. Pictorially, a matrix in Smith normal form looks like

$$A=\begin{bmatrix} a_1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & a_2 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & a_3 & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & a_m & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}$$

For a nonzero matrix $A = (a_{ij})$ with coefficients in R, we let $d(A)$ be the greatest positive number dividing all coefficients, so $d(A) = \gcd\{a_{ij}: 1 \le i \le k, 1 \le j \le k\}$. Let r_i be the i-th row of A and let c_j be the j-th column of A. Recall the row and column operations:
R_1: switches the i-th and j-th row of A.
R_2: multiplies the i-th row of A by -1.
R_3: replaces the i-th row of A by $r_i + ar_j$ for some $a \in R$.
C_1: switches the i-th and j-th columns of A.
C_2: multiplies the i-th column of A by -1.
C_3: replaces the i-th column of A by $c_i + ac_j$ for some $a \in \mathbb{Z}$.

Lemma 1. Let B be the result of applying a row or column operation to A. Then $d(A) = d(B)$.
Proposition 1. Let A be a nonzero matrix with integer coefficients and let $d = d(A)R$. Then we can use row and column operations to transform A into a matrix B with $t(B) = d$.

Theorem 1. If A is a matrix with integer coefficients, we can transform A into Smith normal form using row and column operations.

Example 2.

$$\begin{bmatrix} 2 & 0 & 1 & \cdots & 1 & 0 & 0 \\ 4 & 5 & 2 & \cdots & 0 & 1 & 0 \\ 3 & 7 & 0 & \cdots & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix} \mapsto \begin{bmatrix} 2 & 0 & 1 & \cdots & 1 & 0 & 0 \\ 0 & 5 & 0 & \cdots & -2 & 1 & 0 \\ 1 & 7 & -1 & \cdots & -1 & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix} \mapsto \begin{bmatrix} 0 & -14 & 3 & \cdots & 3 & 0 & -2 \\ 0 & 5 & 0 & \cdots & -2 & 1 & 0 \\ 1 & 7 & -1 & \cdots & -1 & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix}$$

$$\mapsto \begin{bmatrix} 0 & 7 & -1 & \cdots & -1 & 0 & 1 \\ 0 & 5 & 0 & \cdots & -2 & 1 & 0 \\ 0 & -14 & 3 & \cdots & 3 & 0 & -2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 & \cdots & -1 & 0 & 1 \\ 0 & 5 & 0 & \cdots & -2 & 1 & 0 \\ 0 & -14 & 3 & \cdots & 3 & 0 & -2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & -7 & 1 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 & \cdots & -1 & 0 & 1 \\ 0 & 5 & 0 & \cdots & -2 & 1 & 0 \\ 0 & 1 & 3 & \cdots & -3 & 3 & -2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & -7 & 1 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix}$$

$$\mapsto \begin{bmatrix} 1 & 0 & 0 & \cdots & -1 & 0 & 1 \\ 0 & 1 & 3 & \cdots & -3 & 3 & 2 \\ 0 & 0 & -15 & \cdots & 13 & -14 & 10 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & -7 & 1 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 & \cdots & -1 & 0 & 1 \\ 0 & 1 & 0 & \cdots & -3 & 3 & 2 \\ 0 & 0 & -15 & \cdots & 13 & -14 & 10 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & -7 & 22 & \cdots & & & \\ 0 & 1 & 0 & \cdots & & X & \\ 0 & 0 & 1 & \cdots & & & \end{bmatrix}$$

Smith normal form:

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -15 \end{bmatrix}.$$

Rational (Frobenius) Normal Form

Lemma 2. Let $A = k[t]$ and let M be a cyclic torsion A-module (hence, M is finite dimensional as a k vector space). Let $q(t) = t^n + a_{n-1}t^{n-1} + \ldots + a_0$ be the minimal polynomial of t considered as a linear operator on M. Prove that M has a basis with respect to which the matrix for t has the following form:

$$\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -a_{n-1} \end{bmatrix}$$

Theorem 2. Let M be a finitely generated torsion $k[t]$-module. Prove that there exist non-constant monic polynomials $q_1(x), q_2(x), \ldots, q_m(x)$, determined uniquely, such that $q_1(x)|q_2(x)|\ldots|q_m(x)$ and

$$M \cong A/(q_1(x)) \oplus A/(q_2(x)) \oplus \ldots \oplus A/(q_m(x)).$$

Definition 5. The polynomials $q_1(x), q_2(x), \ldots, q_m(x)$, are called the **invariant factors** of M.

Definition 6. Let V be a finite dimensional vector space over k, and let $\mathfrak{I}: V \to V$ be a k-linear operator. Consider a ring homomorphism

$$\phi: k[t] \to End_k(V)$$

defined by sending t to $\mathfrak{I}$ and extending (k)-linearly. Let $Ker\phi$ be the kernel ideal, and let $q_{\mathfrak{I}}(t)$ be a monic polynomial in $k[t]$ which generates $Ker\phi$. Then $q_{\mathfrak{I}}(t)$ is the **minimal polynomial** of $\mathfrak{I}$.

Note that $q_{\mathfrak{I}}(t)$ exists since $k[t]$ is a PID and is unique since we assume it is monic.

Theorem 3. Let V be a finite dimensional k-vector space, and let $\mathfrak{I}: V \to V$ be a k-linear transformation of V. Let $\chi_{\mathfrak{I}}(t)$ be the characteristic polynomial of $\mathfrak{I}$.

• There exist uniquely determined non-constant monic polynomials $q_1(x)$, $q_2(x), \ldots, q_m(x)$, such that
  - $q_i(x)|q_{i+1}(x)$ for $1 \le i \le m-1$;
  - $q_m(x)$ is the minimal polynomial of $\mathfrak{I}$;
  - $\chi_{\mathfrak{I}}(t) = \varepsilon q_1(x)q_2(x)\ldots q_m(x)$ for $\varepsilon \in k$.

• There exists a basis of V such that the matrix of $\mathfrak{I}$ with respect to this basis is a block matrix with m blocks $A_1, A_2, \ldots, A_m$ and each block A_i is in the Frobenius normal form for q_i.

Now, we describe general strategy on how to find a decomposition of a finitely generated abelian group into a direct sum of cyclic subgroups. This strategy can be formalized to give a different, more constructive, proof of the structure theorem.

General strategy: Let M be a finite dimensional A-module, where A is a PID.

Then M fits in a short exact sequence $0 \to R \xrightarrow{T} F \to M \to 0$ where both $F \cong A^n$ and $R \cong A^n$ are free modules. The embedding $R \to F$ is given by a matrix T of size $m \times n$ and rank m. Claim: There exists a change of bases for both R and F, described by the

matrices P and Q respectively such that QTP, the matrix of the embedding $R \to F$ in the new bases, is diagonal and has the form

$$\begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & d_3 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & d_l & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \end{bmatrix}$$

where $d_1|d_2|\ldots|d_l$ are the invariant factors of M, and the number of zero rows (if any) is the rank of M. Then,

$$M \cong A/(d_1) \oplus A/(d_2) \oplus \ldots \oplus A/(d_l) \oplus A^{n-l}.$$

Example 3.

$$A = \begin{bmatrix} -1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$tI - A = \begin{bmatrix} t+1 & -1 & -1 \\ 0 & t & -1 \\ 0 & -1 & t \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 \\ 0 & t+1 & 0 \\ 0 & 0 & t^2-1 \end{bmatrix}$$

$$A \approx \begin{bmatrix} C_{t+1} & 0 \\ 0 & C_{t^2-1} \end{bmatrix} \mapsto \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Hermite Normal Form

We give an efficient algorithm to compute the Hermite normal form of arbitrary integer matrices, and then use it to solve various lattice problems.

Definition 7. *A* non-singular matrix $B = [b_1 \ldots b_n] \in \mathbb{R}^{m \times n}$ is in Hermite normal form (HNF) iff

- There exists $1 \le i_1 \le \ldots \le i_h \le m$ such that $b_{ij} \ne 0 \Rightarrow (i < h) \wedge (i \ge i_j)$ (strictly decreasing column height);
- For all $k > j$, $0 \le b_{i_j,k} < b_{i_j,j}$, i.e., all elements at rows i_j are reduced modulo $b_{i_j,j}$.

The index h is the number of non-zero columns in the matrix, and index i_j is the row of the top non-zero element of column j. Because these terms are strictly increasing, each column contains rows that none of the later columns have. Thus, the non-zero columns of a matrix in HNF (as well as the rows with indices i_j) are guaranteed to be

linearly independent. We also know (but will not show here) that HNF is unique, i.e., if two matrices B and B' are in HNF and they generate the same lattice ($\Im(B) = \Im(B')$), then $B = B'$ (except at most for the number of zero-columns at the end). We say that H is the HNF of B if $\Im(H) = \Im(B)$, H is in HNF, and H does not contain zero columns. It is not hard to come up with an algorith that computes the HNF of a matrix performing only a polynomial number of operations. However, a naive solution to this problem may result in superpolynomial running time because the numbers in the intermediate computations can easily get very large. In order to achieve polynomial running time, some care is required.

Notice that the problem of computing the HNF of a rational matrix $B \in Q^{m\times n}$ easily reduces to the problem of computing the HNF of an integer matrix as follows:

- let D be the least common multiple of all denominators occurring in B;
- Compute the HNF H of integer matrix $D \cdot B \in Z^{m\times n}$;
- Output $D^{-1} \cdot H$.

It is easy to verify that that H is in HNF if and only if $D \cdot B$ is. Moreover, if H and $D \cdot B$, generate the same lattice, then $D^{-1} \cdot H$ and B also generate the same lattice. Therefore, $D^{-1} \cdot H$ is the HNF of B. This reduction is polynomial time because $\log_2 D$ is at most as big as the bitsize of the input matrix. In the rest of this section we show how to compute the HNF of an integer matrix.

So, we may assume that the input matrix has integer entries. We first give an algorithm to compute the HNF of matrices with full row rank, and then show how to adapt it to arbitrary matrices.

Matrices with full row rank. We give an algorithm that can find the HNF of any matrix $B \in Q^{m\times n}$ which has full row rank. We know that in this case the HNF of B is a square non-singular $m \times m$ matrix H. The idea is to first find the HNF basis H of a sublattice of $\Im(B)$, and then update H by including the columns of B one by one. Let's assume for now that we have a polynomial time procedure AddColumn that on input a square non-singular HNF matrix $H \in Z^{m\times n}$ and a vector boutputs the HNF of matrix $[H|b]$. The HNF of B can be computed as follows:

- Apply the Gram-Schmidt algorithm to the columns of B to find m linearly independent columns. Let B' be the $m \times m$ matrix given by these columns;
- Compute $d = \det(B')$, using the Gram-Schmidt algorithm or any other polynomial time procedure. Let $H_0 = d \cdot I$ be the diagonal matrix with d on the diagonal.
- For $i = 1, \ldots, n$ let H_i the result of applying AddColumn to input H_{i-1} and b_i.
- Output H_n.

The correctness of the algorithm is based on the invariant that for all i, H_i is the HNF of the lattice $\Im([d \cdot I|b_1, \ldots, b_i])$. The invariant is clearly satisfied for $i = 0$. Moreover, it is preserved at every iteration by definition of AddColumn. So, upon termination, the algorithm outputs the HNF of $\Im([d \cdot I|B])$. Finally, since $d \cdot Z^m \subseteq \Im(B') \subseteq \Im(B)$, we have $\Im([d \cdot I|B]) = \Im(B)$ and the algorithm outputs the HNF of B.

Notice that during the entire process, all the entries of H_i stay bounded by d. In particular, all the numbers are polynomial in the original input B. So, if AddColumn

is polynomial time, then the entire HNF algorithm is polynomial time. It remains to give an algorithm for the AddColumn procedure. On input a square non-singular HNF matrix $H \in \mathbb{Z}^{m\times n}$ and a vector $b \in \mathbb{Z}^m$, add column proceeds as follows. If $m = 0$, then there is nothing to do, and we can immediately terminate witutput H. Otherwise, let

$$H = \begin{bmatrix} a & 0^T \\ h & H' \end{bmatrix}$$

and

$$b = \begin{bmatrix} b \\ b' \end{bmatrix}$$

and proceed as follows:

• Compute $g = \gcd(a, b)$ and integers x, y such that $xa + yb = g$ using the extended gcd algorithm;

• Apply the unimodular transformation

$$U = \begin{bmatrix} x & -b/y \\ y & a/g \end{bmatrix}$$

to the first column of H and b to obtain

$$\begin{bmatrix} a & b \\ h & b' \end{bmatrix} U = \begin{bmatrix} g & 0 \\ h' & b'' \end{bmatrix}$$

• Add an appropriate vector from $\Im(H')$ to b'' so to reduce its entries modulo the diagonal elements of $H.'$

• Recursively invoke AddColumn on input H' and b'' to obtain a matrix H''

• Add an appropriate vector from $\Im(H'')$ to h' so to reduce its entries modulo the diagonal elements of H''.

• Output

$$\begin{bmatrix} g & 0^T \\ h' & H'' \end{bmatrix}$$

General case. We would like to reduce the general case to the full-dimensional case. We begin by using a projection operation Π to select B', a submatrix of B consisting only of linearly independent rows of B. Then we use the algorithm for the full-dimensional case. Finally, we use the inverse of the projection operation to get our final result.

• Run the Gram-Schmidt orthogonalization process on the rows $r_1, \ldots, r_m$ of B, and let $K = \{k_1, \ldots, k_l\}$ $(k_1 < \ldots < k_l)$ be the set of indices such that $r^*_{k_i} \neq 0$. Define the projection operation $\Pi_K : R^m \to R^l$ by $[\Pi_K(x)]_i = x_{k_i}$. Notice that the rows $r_k (k \in K)$ are linearly independent and any other row can be expressed as a linear combination of the previous rows $r_j (j \in K : j < i)$. Therefore Π_K is one-to-one when restricted to $\Im(B)$, and its inverse can be easily computed using the Gram-Schmidt coefficients $\mu_{i,j}$.

• Define a new matrix $B' = \Pi_K(B)$, which is full-rank, and run the algorithm given in the previous section to find the HNFB'' of B'.

• Apply the inverse projection function, Π_K^{-1}, to the HNFdetermined in the previous step B'', to give matrix H. It is easy to see that $\Im(H)\Im(B)$ and H is in HNF. Therefore H is the HNF of B.

Example 4.

$$\begin{bmatrix} 2 & 3 & 6 \\ 2 & 1 & -3 \\ 1 & 1 & 3 \end{bmatrix} \mapsto \begin{bmatrix} 2 & 3 & 0 \\ 2 & 1 & -9 \\ 1 & 1 & 0 \end{bmatrix} \mapsto \begin{bmatrix} 2 & 1 & 0 \\ 2 & -1 & -9 \\ 1 & 0 & 0 \end{bmatrix} \mapsto \begin{bmatrix} 0 & 1 & 0 \\ 4 & -1 & -9 \\ 1 & 0 & 0 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 \\ -1 & 4 & -9 \\ 0 & 1 & 0 \end{bmatrix}.$$

Python code

```
from tkinter import *
import tkinter as tk
from smithnormalform import matrix, snfproblem, z
from sympy import Matrix
import numpy as np
from hsnf import column_style_hermite_normal_form,
row_style_hermite_normal_form, smith_normal_form

def dodajPoljaZaElementeMatrice():
    global l
    global m
    global n
    but_1["state"] = DISABLED
    ent_m["state"] = DISABLED
    ent_n["state"] = DISABLED
    m=int(ent_m.get())
    n=int(ent_n.get())

    l=[]

    for i in range(m):
        for j in range(n):
            l.append(None)
    br=-1
    for i in range(m):
        for j in range(n):
            br=br+1
            entry_text = tk.StringVar()
            l[br] = Entry(prozor,width=5,textvariable=entry_text)
            l[br].place(x=50+j*50, y=200+i*50)
            entry_text.set(str(br+1))
    but_2 = Button(prozor, text = "Smith Normal Form",width=40,
```

```
        command = Smith_Normal_Form)
    but_2.place(x=0, y=250+m*50)

    but_3 = Button(prozor, text = "Jordan Normal Form",width=40,
        command = Jordan_Normal_Form)
    but_3.place(x=0, y=300+m*50)

    but_4 = Button(prozor, text = "Hermite Normal Form",width=40,
        command = Hermite_Normal_Form)
    but_4.place(x=0, y=350+m*50)

def Hermite_Normal_Form():
    global l
    global m
    global n
    x=[]
    for el in l:
       x.append(el.get())
    a=np.array(x,dtype='int32')
    M=a.reshape(m, n)

    t4 = Text(prozor,width=30)
    t4.insert(END, "Matrica M:\n")
    t4.insert(END, M)
    # Row-style hermite normal form
    H, L = row_style_hermite_normal_form(M)
    """
    H = array([
    [   1    0   420  -2522]
    [   0    3  1809 -10860]
    [   0    0  2079 -12474]])
    """
    assert np.allclose(L @ M, H)
    assert np.around(np.abs(np.linalg.det(L))) == 1  # unimodular

    t4.insert(END, "\nMatrica H:\n")
    t4.insert(END, H)
    t4.insert(END, "\nMatrica L:\n")
    t4.insert(END, L)

    # Column-style hermite normal form
    H, R = column_style_hermite_normal_form(M)
    """
    H = array([
```

```
    [  3   0   0   0]
    [  0   1   0   0]
    [1185  474 2079   0]])
    """
    assert np.allclose(np.dot(M, R), H)
    assert np.around(np.abs(np.linalg.det(R))) == 1  # unimodular

    t4.insert(END, "\nMatrica H:\n")
    t4.insert(END, H)
    t4.insert(END, "\nMatrica R:\n")
    t4.insert(END, R)

    t4.place(x=0, y=450+m*50)
def Jordan_Normal_Form():
    global l
    global m
    global n
    x=[]
    for el in l:
        x.append(el.get())
    a=np.array(x)
    a=a.reshape(m, n)
    A = Matrix(a)
    P, J = A.jordan_form()
    '''
    print("Matrica M:")
    print(A)
    print("Matrica J:")
    print(J)
    print("Matrica P:")
    print(P)
    '''
    t2 = Text(prozor,width=100)
    t2.insert(END, "Matrica A:\n")
    for i in range(m):
        t2.insert(END,  A[i,:][:])
        t2.insert(END, "\n")
    t2.insert(END, "Matrica J:\n")
    for i in range(m):
        t2.insert(END,  J[i,:][:])
        t2.insert(END, "\n")
    t2.insert(END, "\nMatrica P:\n")
    for i in range(m):
        t2.insert(END,  P[i,:][:])
```

```
        t2.insert(END, "\n")
    t2.place(x=0, y=450+m*50)

def Smith_Normal_Form():
    global l
    global m
    global n
    x=[]
    for el in l:
        x.append(z.Z(int(el.get())))
    original_matrix = matrix.Matrix(m, n, x)
    prob = snfproblem.SNFProblem(original_matrix)
    prob.computeSNF()
    '''
    print(prob.isValid())
    print(prob.A)
    print(type(prob.A))

    print(prob.J)
    print(prob.S * prob.A * prob.T == prob.J)
    '''
    t1 = Text(prozor,width=30)
    t1.insert(END, "Matrica A:\n")
    t1.insert(END, prob.A)
    t1.insert(END, "Matrica J:\n")
    t1.insert(END, prob.J)

    t1.place(x=0, y=450+m*50)

prozor = Tk()

prozor.geometry("1000x1000")
prozor.configure(bg='lightgreen')
common_img = PhotoImage(width = 1, height = 1)
l=[]
#unos vrednsoti za broj vrsta m
lab_m = Label(prozor, text = "broj vrsta m=")
lab_m.place(x=0,y=0)

m=IntVar(prozor)
ent_m = Entry(prozor,textvariable=m)
ent_m.place(x=100,y=0)
```

```
#unos vrednsoti za broj kolona n
lab_n = Label(prozor, text = "broj kolona n=")
lab_n.place(x=0,y=50)

n=IntVar(prozor)
ent_n = Entry(prozor,textvariable=n)
ent_n.place(x=100,y=50)

but_1 = Button(prozor, text = "Dodaj Polja Za Elemente Matrice",
          command = dodajPoljaZaElementeMatrice)
but_1.place(x=0,y=150)

prozor.mainloop()

print('kraj')
```

Conclusion

For each of these four normal forms, algorithms are written in the Python programming language. The row of the matrix is limited to power 3.

It follows from Hermitc that any m × n rank n integer matrix d can be transformed by applying a sequence of integer row operations to an upper triangular matrix H that has off-diagonal entries nonnegative and with magnitude smaller than the positive diagonal entry in the same column. The triangularization H - called the Hermite normal form of A - always exists and is unique. In this paper we consider the problem of computing the Hermite, Jordan, Smith and rational normal form of an n x n non-singular matrix T that is already upper triangular and has offdiagonal entries bounded in magnitude by the product D of the diagonal entries.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974.

[2] E. H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Mathematics of Computation*, 22(103): 565–578, 1968.

[3] E. H. Bareiss. Computational solution of matrix problems over an integral domain. *Phil. Trans. Roy. Soc. London*, 10:68–104, 1972.

[4] S. Barnette, I. S. Pace. Efficient algorithms for linear system calculation; part I — Smith form and common divisor of polynomial matrices. *Internat. J. of Systems Sci.*, 5: 403–411, 1974.

[5] W. A. Blankinship. Algorithm 287, Matrix triangulation with integer arithmetic. *Communications of the ACM*, 9(7): 513, July 1966.

[6] W. A. Blankinship. Algorithm 288, solution of simultaneous linear diophantine equations. *Communications of the ACM*, 9(7): 514, July 1966.

[7] G. H. Bradley. Algorithms for Hermite and Smith normal form matrices and linear diophantine equations. Mathematics of Computation, 25(116):897–907, Oct. 1971.
[8] J. Cannon, G. Havas. Algorithms for groups. *Australian Computer Journal*, 24(2):51–58, May 1992.
[9] B. W. Char, K. O. Geddes, G. H. Gonnet. GCDHEU: Heuristic polynomial GCD algorithm based on integer GCD computations. *Journal of Symbolic Computation*, 7(1):31–48, Jan. 1989.
[10] P. D. Domich, R. Kannan, L. E. Trotter, Jr. Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research*, 12(1):50–59, Feb. 1987.
[11] M. A. Frumkin. An application of modular arithmetic to the constuction of algorithms for solving systems of linear equations. *Soviet Math. Dokl*. 17:1165–1168, 1976.
[12] F. R. Gantmacher. Matrix Theory, volume 1. Chelsea Publishing Company, 1960.
[13] K. O. Geddes, S. R. Czapor, and G. Labahn. Algorithms for Computer Algebra. Kluwer, Boston, MA, 1992.
[14] B. Hartley and T. O. Hawkes. Rings, Modules, and Linear Algebra. Chapman and Hall, 1970. [18] G. Havas, D. F. Holt, and S. Rees. Recognizing badly presented Z-modules. *Linear Algebra and its Applications*, 192:137–163, 1993
[15] O. Zariski, Pierre Samuel. *Commutative Algebra*. Van Nostrand, Princeton, N.J., 1958–1960
[16] P. Robbins, *Python Progamming for Beginners*, 2023
[17] T. Theis, Getting Started with Python, 2024