Smith normal form - Programming final project

Louis Philippe Ignatieff, Qiu Shi Wang and Jing Han Sun

May 28, 2020

1 User guide

We state the main result motivating our project.

Theorem 1 (Smith Normal Form)**.** *Let $A \in \mathrm{Mat}_{m\times n}(\mathbb{Z})$ be a matrix with integer entries. Then there exist integer matrices $U_{m\times m}, D_{m\times n}$ and $V_{n\times n}$ such that:*

- *$D = UAV$ is called the* ***Smith normal form*** *of A.*
- *D is a diagonal matrix, that is, $D_{ij} = 0$ whenever $i \neq j$.*
- *Each diagonal entry of D divides the next, that is, $D_{ii}|D_{i+1,i+1}$.*
- *U and V are invertible over $\mathbb{Z}$.*

Proof. The existence of the Smith normal form is proven by construction, by calculating it using the algorithm implemented in the program. It is also true that the Smith normal form and the matrices U and V are unique up to signs. We will not reproduce the proof here, it is in Hoffman and Kunze (1971) $\square$

The nonzero entries D_{ii} are called the *elementary divisors* of A; they are unique up to signs. Also note that the last item is equivalent to saying that U and V have determinant ± 1.

There are three .py files that can be used: smithcalc.py, homology.py and similarity.py.

Smith normal form calculator: smithcalc.py

Run smithcalc.py with Python by double-clicking it. You will be prompted to enter an integer matrix, and the program will output the matrices D, U and V, in this order. D is the Smith normal form.

Homology calculator: homology.py

An important application of the Smith normal form is in the calculation of homology groups of chain complexes of finitely generated free abelian groups. In fact, it allows us to calculate quotient groups of free abelian groups, which are abelian but not necessarily free. Using the fact that the much simpler $\mathrm{im}(D)$ is isomorphic to $\mathrm{im}(A)$ since U is invertible, we can calculate the torsion direct summands of the homology group in terms of the elementary divisors of A. We calculate the free direct summand from the dimensions of $\ker(B)$ and $\mathrm{im}(A)$, obtained from the sizes and ranks of the matrices. We then have the following:

Theorem 2. *For the chain complex $\mathbb{Z}^n \xrightarrow{A} \mathbb{Z}^m \xrightarrow{B} \mathbb{Z}^k$ ($BA = 0$ by definition of a chain complex), the homology group at the middle term is*

$$\frac{\ker(B)}{\mathrm{im}(A)} \cong \bigoplus_{i=1}^{r} \mathbb{Z}/a_i \oplus \mathbb{Z}^{m-\mathrm{rank}(A)-\mathrm{rank}(B)}$$

Where $\mathbb{Z}/p$ is the integers modulo p, and $a_1, ..., a_r$ are the nonzero elementary divisors of A.

Thus we can use our algorithm to calculate homology groups. For example, we can calculate the homology of a cellular chain complex

$$\cdots \to H_{n+1}(X^{n+1}, X^n) \to H_n(X^n, X^{n-1}) \to H_{n-1}(X^{n-1}, X^{n-2}) \to \cdots$$

since for all n the groups $H_n(X^n, X^{n-1})$ are free abelian generated by the n-cells of X. The matrices A and B are calculated using the degrees $d_{\alpha\beta}$ of the composition of the attaching map of the cell e^n_α and the quotient map sending all $(n-1)$-cells except e^{n-1}_β to zero.

To use the program, run the file homology.py and enter the two integer matrices A and B with a comma separating them. The program will then print out the homology group.

Matrix similarity test: similarity.py

Another application of the Smith normal form is that it allows us to determine whether two square matrices over $\mathbb{Q}$ are similar. Recall that two matrices A and B are similar if there exists an invertible matrix P such that $A = PBP^{-1}$. We state without proof the result that allows this:

Theorem 3. *Two matrices A_{nn} and B_{nn} are similar if and only if the Smith normal forms of their characteristic matrices $xI - A$ and $xI - B$, as polynomial matrices, are equal.*

To use the program, run the file similarity.py and enter two integer matrices A and B as in the entry for homology.py. The program will then determine whether they are similar over $\mathbb{Q}$. If the entries are not integers but fractions, multiply them by the least common multiple of the denominators then put them in the program.

2 Design guide

The Smith normal form algorithm

The main document, smith.py, contains the smith() method as well as three independent methods exchange_rows(), RRcol() which are related to row-reduction and egcd(), the extended Euclidean algorithm, which all serve to compute the Smith normal form.

The algorithm has many steps. Steps 1 to 4 are repeated once for each value of t from 1 to the number of rows in A.

Step 1 chooses a pivot for the following steps. This is the first non-zero entry of the first non-zero column. We bring this entry to the t-th row of the matrix using exchange_rows().

Step 2 improves the pivot only if entries on its rows or columns are not multiples of the pivot by computing the greatest common divisor and coefficients satisfying Bézout's identity using egcd(), an implementation of the extended Euclidean algorithm that we found online. egcd() is lines 216 to 222 of smith.py and is the only code in the whole project that we did not write.

Step 3 consists of using RRcol() to row-reduce the column entries under the pivot to zeroes. This is possible within the integers since Step 2 made all the entries under the pivot into multiples of the pivot.

For Step 4, we transpose the matrix and repeat the operations of Step 2 and Step 3 for the row of the pivot, that way, we do not need to implement column operations.

Since in general, Step 2 results in non-zero entries in the row of the pivot, we need to iterate over Steps 2 and 3 until Step 2 is skipped entirely. There is a mathematical proof, beyond the scope of our project, that the while loop which iterates Steps 2 and 3 terminates eventually.

After Steps 1 to 4, there should be at most one entry in each row and column.
Step 5 pushes all empty columns to the right end of the matrix. The matrix should then be diagonal where all non-zero entries are at the top left.

Step 6 ensures that every entry on the diagonal is a multiple of the previous entry. We accomplish this by using row operations on the entries of the diagonals and their right neighbour and compute their greatest common divisor as the new entry of the diagonal.

Step 7 ensures that the entries of the Smith normal form are positive, as by convention.

All operations realized on the original matrix are elementary operations which are invertible over $\mathbb{Z}$, and hence we construct the right and left invertible integer matrices U, and V as a product of the elementary matrices. They are collected in the rightlist and leftlist, and multiplied at the end.

Our implementation of a class of matrices

In addition to the smith normal form algorithm, we decided to implement a rudimentary matrix class in Matrix1.py which we based mostly on our experience of the sympy matrix class methods. This class defines a matrix from a list of lists, where the inner lists are rows, with attributes ncols, nrows and shape. We assume the rows of the matrix all have the same length by the fact that it is a matrix.

Using the properties and methods of lists, we construct the matrix methods needed, with mutable entries. Indices, assignments, equality, inequality addition, subtraction, rows and columns, and more interestingly, transpose, scalar multiplication and matrix multiplication. We construct the zero and identity matrices as global functions outside of the class. With all these, we can proceed with the Smith normal form algorithm.

We also implement matrix row operations, Gauss-Jordan row reduction and rank. These are only necessary for our applications to homology calculations and matrix similarity testing.

Homology calculator

The implementation of this calculator is straightforward as it simply takes the nonzero elements of the Smith normal form of A (the elementary divisors of A) and prints them in the coefficients p of finite cyclic groups $\mathbb{Z}/p$. The code is somewhat lengthy due to the special cases required to print an answer that is a complete and legible English sentence, and also to manually remove the trivial cases of $\mathbb{Z}^0$ and $\mathbb{Z}/1$, both isomorphic to the trivial group. The algorithm requires rank() implemented in Matrix1.py to calculate the free part of the group.

Similarity calculator

The issue with testing whether two matrices are similar is that we cannot calculate the Smith normal form of a matrix with polynomial entries, since our Matrix1 method only supports integers and floats. However, a workaround exists: we simply compare the Smith normal forms of the characteristic matrices of A and B for different values of x. For A and B of size n, the entries of their Smith normal forms are polynomials of degree at most n. Therefore, by testing $n+1$ distinct x values, we can show equality between the two polynomial matrices. For technical reasons, we choose values of x such that the characteristic matrices of A and B are nonsingular.