Modular HNF

Loïc Grenié

26 Janvier 2012

1 HNF

Definition 1 *Let H be an matrix with integer coefficients. The matrix H is said to be in* HNF form *if*

$$H = \begin{pmatrix} z_1 & c_1 & a_{12} & \dots & a_{1k} \\ z_2 & & c_2 & \dots & a_{2k} \\ \vdots & & & \ddots & \vdots \\ z_k & & & & c_k \end{pmatrix}$$

where each c_i is a column with positive bottom coefficient p_i and on the right of each p_i each coefficient x of the matrix satisfy $0 \leqslant x < p_i$ and all z_i are zero matrices (possibly of 0 width).

Theorem 2 *Let M be an integer matrix. There exists an integer matrix H in HNF form and an integer matrix U with determinant ± 1 such that*

$$H = MU \ .$$

Moreover H is uniquely determined by M while U is determined only up to the kernel of M.

2 The state of pari

We need the pari (H, U) of the theorem above for various reasons. We will speak of one of them tomorrow.

There are at least two implementations of the HNF algorithm right now. The first one, which we will call naïve, can be used under GP using `HU=mathnf(M,1);` The result is a two component vector `HU`, where `HU[1]` is H and `HU[2]` is U. The second algorithm, which we will call LLL, can be used with `HU=mathnf(M,4);` with the same output. The naïve algorithm is rather fast for small input but the matrix U is much worse than the LLL one; moreover the naïve algorithm does not always finish even with reasonably small input. The LLL algorithm is slower but always (barring bugs) finish and gives a nearly optimal U.

For the applications, it is usually good to have a matrix U with as small an L^2-norm as possible. This means that LLL algorithm is usually the best one, but it is relatively slow. We will discuss a different algorithm that outputs a good U but faster than the LLL algorithm.

3 The algorithm

Let M be an $m \times n$ integer matrix. For simplicity, we suppose $\operatorname{rk} M = m$. That way $k = m$ in the definition above and each c_i is the 1×1 matrix (p_i) and we identify them.

The algorithm to compute (H, U) has several steps, separated in two phases.

3.1 Computing HNF

The first phase is computing the HNF H and a matrix U such that

$$H = MU \ .$$

3.1.1 Wrong result

We first compute an integer matrix U_1 such that $H_1 = MU_1$ is a matrix $H_1 = (Z \mid dI_m)$ with Z a null $m \times (n-m)$ matrix and $d > 0$. The matrix H_1 is better than HNF however U_1 does (definitely) not have determinant ± 1.

Precisely, we compute modulo a certain number of primes p matrices $U_{1,p}$ such that

$$MU_{1,p} \equiv (Z \mid d_p I_m) \pmod{p}$$

where $d_p \neq 0$ is the modulo p determinant of an $m \times m$ submatrix of M. We compute d_p and the submatrix using Gauss algorithm (modulo p) and we make sure that we extract the same submatrix modulo all the primes p. Precisely, if one of the pivot we have found for a previous prime is zero modulo some prime p we just discard prime p and if we find a new pivot modulo p we discard the $U_{1,\ell}$ and d_ℓ we had computed so far.

We compute the matrix U_1 and d by chinese remainder theorem and stop whenever the matrix $H_1 = MU_1$ has the desired shape. At that point, the number d is the determinant of the submatrix we extracted and the matrix U_1 has determinant d^{n-1}.

3.1.2 Look for primes

We factor the determinant d using the polynomial time factorization algorithm.

3.1.3 Correct U

We set $U_2 = U_1$ and $H_2 = H_1$. We will denote $A[k]$ the k-th column of any matrix A.

For each prime p dividing d, we compute the kernel of the matrix U_2 modulo p. If kernel is trivial, skip to the next prime. Otherwise, PARI provides us with a basis of the kernel made of vectors of the form

$$\begin{pmatrix} a_1 \\ \vdots \\ a_{k-1} \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This means that $U_2[k] + \sum_{i=1}^{k-1} a_i U_2[i]$ is divisible by p. We thus substitute

$$\frac{1}{p}\left(U_2[k] + \sum_{i=1}^{k-1} a_i U_2[i]\right)$$

for $U_2[k]$. This has the effect of dividing the determinant of U_2 by p. At the same time we update $H_2[k]$ by the similar formula. The k-th column of H_2 will have non-zero coefficients only in the first k components because we use only lower numbered columns to modify H_2. This means that H_2 remains in nearly-HNF form (the coefficients at the right of a pivot can be larger than the pivot). We go on computing the kernel modulo p until it becomes trivial (which will be happend when the determinant of U_2 is not divisible by p).

3.1.4 Correct H

We set $H = H_2$ and cleanup H so that the elements on the right of the pivots become smaller than the pivot.

3.2 Optimize U

The four steps above do not give a very good result in terms of the L^2-norm of U. If the matrix M has a trivial kernel there is only one choice for U so that nothing can be done. However if M has a non-trivial kernel, we can modify each column of U by any vector of the kernel. We want to find the best choice to have the smallest possible vectors in U.

Let r be the dimension of the kernel of M and let K be the $\mathbf{Z}$-module generated by the r left-most columns of U. The r left-most columns of H are zero columns (and the $n - r$ right-most ones are not).

3.2.1 LLL kernel

We begin by computing an LLL-reduced basis $(v_1, ..., v_r)$ of K and we substitute the r first columns of U by this LLL-reduced basis.

3.2.2 Babai on the other columns

For k such that $r < k \leqslant n$, we use Babai algorithm to compute the element v_k of K nearest to $U[k]$. We then substract v_k to $U[k]$.

3.2.3 Gram-Schmidt

We can further optimize the vectors by using Gram-Schmidt technique. If $1 \leqslant k \leqslant r$ and $1 \leqslant i \leqslant n$ (with $k \neq i$) we can substract

$$\left[\frac{(U[k] \mid U[i])}{(U[k] \mid U[k])}\right] U[k]$$

to $U[i]$. The coefficient of $U[k]$ is not garanteed to be 0 however it is most of the time and is never very large. This optimization can be done (with $i \leqslant r$) before using Babai algorithm as well as after (with $i > r$).