[Tutorial]

S-units and compact representations in number fields

Karim Belabas

Institut de
Mathématiques de
Bordeaux

Use case / motivation

```
?  T = x^6 + 2854*x^4 + 2036329*x^2 + 513996528;
?  K = bnfinit(T);   \\
```
$K = \mathbb{Q}[x]/(T)$, require class group and units

```
?  K.fu    \\
```
missing units

```
% = 0
?  K = bnfinit(T, 1);    \\
```
impose units computation

```
?  K.fu
```

\\ . . . *huge* result deleted . . .

Huge algebraic numbers are problematic because

- computing with them algebraically is expensive;
- approximations via floating point embeddings into $\mathbb{C}$ require huge accuracy (cancellation);
- they are often intermediate results: we do not want a result in K but in $K^*/(K^*)^2$, or in $\mathbb{Z}_K/\mathfrak{p}^k$, or a floating point approximation to complex embeddings, or . . .
- they may overflow the possibilities of the implementation: try $2^{2^{100}}$.

Number field structures

Let $K = \mathbb{Q}[x]/(T)$ be a number field of degree $[K : \mathbb{Q}] = n$; let

- (r_1, r_2) be its signature, S_∞ be the set of r_1 real places and r_2 complex places,
- $\mathbb{Z}_K = \mathbb{Z} \cdot b_1 \oplus \cdots \oplus \mathbb{Z} \cdot b_n$ be its ring of integers and d_K its absolute discriminant,
- $\mathrm{Cl}(K)$ be its ideal class group,
- $U(K) = \mathbb{Z}_K^* \sim (\mathbb{Z}/w\mathbb{Z}) \cdot \zeta_w \bigoplus \mathbb{Z} \cdot u_1 \cdots \oplus \mathbb{Z} \cdot u_{r_1+r_2-1}$ be its unit group,
- For $S = S_0 \cup S_\infty$ a finite set of places, let

  $$U_S(K) = \{x \in K^*, v(S) = 0 \text{ for all } v \notin S\}$$

  be the S-unit group; i.e., $U_{S_\infty}(K) = U(K)$. The abelian group $U_S(K)$ is generated by ζ_w and $\#S - 1$ elements of infinite order.

In PARI-speak

- `K = nfinit(T)` allows `K.pol`, `K.sign`, `K.zk`, `K.disc`, `K.p` (ramified primes), ...
- `K = bnfinit(T)` further allows `K.clgp`, `K.tu` (w, ζ_w), `K.fu` (fundamental units), ...

Number field elements

Elements of K are given as

- elements of $\mathbb{Q}$ (rational form): `2`, `1/3`, …
- polynomials (algebraic form): $\texttt{Mod}(1+x, T)$, or simply $1+x$ (implicitly modulo T), …
- vectors (basis form): $[a_1, \ldots, a_n]$~ for $\sum_i a_i w_i$, …

These formats are recognized as inputs by all functions handling algebraic numbers as number field elements. The preferred output format are *rational* and *basis* form, in this order.

```
?  K = nfinit(x^3 - 2);
?  nfeltmul(K, x, x^2+1)
?  nfelttrace(K, x+1)
?  nfeltadd(K, x/2, [1,2,3] )
?  nfbasistoalg(K, %)
?  nfalgtobasis(K, %)
```

Other lossy representations

For the record, let us mention

- chinese remainders (`idealchinese`), including sign conditions at real embeddings;
- projection to residue fields at maximal ideals (`nfmodpr`);
- complex embeddings (`nfeltembed`, floating point);
- projections to more general finite rings $(\mathbb{Z}_K/\mathfrak{f})^*$, $\mathfrak{f} = \mathfrak{f}_0\mathfrak{f}_\infty$ (`ideallog`);
- reduction in $K^*/(K^*)^n$ (`idealredmodpower`).
- factorization into maximal ideals (`idealfactor`), up to units;

These representations alleviate coefficient explosion: they reduce the size of objects and/or the cost of handling them. But they all lose information.

NEW: Compact / factored representation (1/2)

In multiplicative contexts, an element of the form $\prod_i g_i^{e_i}$, where $g_i \in K^*$ and $e_i \in \mathbb{Z}$, can now be represented by a factorization matrix.

We do not have a UFD: the (g_i) need not be coprime! The goal is twofold:

- *avoid coefficient explosion*, measured by the size of the internal representation: compare $2^{1000} \cdot 3^{-2000}$ with its expanded form.
- reduce costs of operations *in multiplicative contexts*: it is easy to multiply or divide formally such objects, reduce modulo squares or larger powers, compute valuations, etc. More generally apply group morphisms $(K^*, \times) \to G$.

NEW: Compact / factored representation (2/2)

There are drawbacks:

- non-multiplicative operations remain expensive, for instance to perform addition we must expand the products first;
- some of them lose useful properties, for instant equality testing: a fast probabilistic algorithm proves that $\prod g_i^{e_i} \neq 1$, but it is hard to prove equality (the g_i are not coprime); failing to disprove equality, we may *assume* equality but we lose guarantees for later steps.
- non-generic simplifications are not taken into account: when expanded outputs *are* small, factored representations are likely to be *larger*;
- backward compatibility !

What does it change ? How to use it ? (1/2)

High level functions transparently use the mechanism behind the scenes (`bnrclassfield`, `bnrstark`, `bnflog`, `thue...`), whenever units or class group computations arise.

- By default, when handling a `bnf = bnfinit`(T) provided by the user, this strategy is less efficient that it could be. It can fail because `bnf` contains floating point data that may not always allow exact algebraic reconstructions. It may also contain huge units in expanded form that contaminate later constructions.
- `bnf = bnfinit`$(T, 1)$ makes the strategy foolproof for that `bnf`, by computing all data in exact algebraic form, using factored representations. Drawback: uses much more memory, and is slower in the worst case although this is not noticeable on average in our tests.

We advise to use `bnfinit(,1)` for all computations and only disable it when it causes `bnfinit` to run into problems.

What does it change ? How to use it ? (2/2)

Caveats / compatibility:

- `bnf.fu` is specified to return units in expanded form. So use the new `bnfunits` instead, which returns units in factored form (and extra information for `bnfisunit`).
- `bnfisprincipal` is specified to return principal ideals in expanded form. So use the new `bnfisprincipal(,4)` flag.

Example: some random real quadratic field. Try these snippets with `bnfinit` instead of `bnfinit(,1)`.

```
D = 1000001273;
K = bnfinit(x^2 - D, 1);
bnfunits(K)
K.fu

P = idealprimedec(K,2)[1];
bnfisprincipal(K, P)
bnfisprincipal(K, P, 4)     \\ factored representation
bnfisprincipal(K, P, 3)     \\ expanded; no longer do this !
```

Addendum: S-units

`bnfunits` also allows to work with general S-units (together with `bnfisunit`). The functions `bnfsunit` and `bnfissunit` are now deprecated.

```
S = idealprimedec(K,2);
U = bnfunits(K, S)
bnfisunit(K, 2)       \\ not a unit
bnfisunit(K, 2, U)     \\ ...but an S-unit
```

Using compact / factored representation

Functions in multiplicative context work "out of the box" with factored representations.

```
? K = nfinit(x^3 - 2);
? u = [x, 2; [1,2,3]~,-1];
? v = [x+1, 1; [-1,2,3]~,2];
? nffactorback(K, u)
%4 = [32/89, 2/89, -11/89]~
? nfeltmul(K, u, v)
%5 =
[           x  2]
[     [1, 2, 3] -1]
[       x + 1  1]
[ [-1, 2, 3]~  2]
```

Using compact / factored representation

```
?  nfeltpow(K, u, 2)
%6 =
 [             x   4]

 [  [1, 2, 3]~  -2]
?  nfeltdiv(K, u, 2)
%7 =
 [             x   2]

 [  [1, 2, 3]~  -1]

 [             2  -1]
?  nfeltnorm(K, u)
%8 = 4/89
```

Using compact / factored representation

```
?  nffactorback(K, [u,v], [2,3])    \\ still factored
%9 =
 [             x   4]

 [   [1, 2, 3]~  -2]

 [         x + 1   3]

 [  [-1, 2, 3]~   6]
?  nffactorback(K, %)    \\ now expand completely
%10 = [93209292/7921, 57744198/7921, 25490430/7921]~
?  nfelttrace(K, u)    \\ not multiplicative ! Fails ...
?  P = idealprimedec(K,5)[2]; nfmodpr(K, v, P)
%11 = 3*x + 3
?  bid = idealstar(K, 5); ideallog(K, v, bid)
%12 = [7, 0]~
```