The modal integer factor of residue *r* (mod *m*)

The *i* of (i;u)-factorization

This algorithm is based on what I judge to be my nicest piece of tight modular reasoning in the year 2019. It starts with the residue r of interest and teases out its modal integer factor i .

Underlying the method is the idea of the **prime factorizations** of m, r, and i — m is a product of prime powers p^k, r of corresponding prime power p^j. If $j \geq k$ then p^j is part of the modal unit of r and is to not to remain a factor of i, otherwise $j < k$ and p^j is a modal integer and should remain a factor of i.

The algorithm

$c_1 \leftarrow \gcd(\, r\,,\, m\,)$ // c_1 is the center of the cluster C_r obtained by factoring out all the euler units
// and clipping the larger exponents of r to their values in m.

$c_2 \leftarrow \gcd(\, c_1^2\,,\, m)$ // c_2 is the center of the cluster one level up from C_r .

The intended result i of this algorithm is that it should have the same modal integers as c_1 but it should have none of the modal units that remain in c_1.

$i \leftarrow c_2/c_1$ // This first step towards i makes it a modal integer (by cancelling out the non-euler
// modal unit factors) but the individual prime powers may have exponents smaller
// than present in the original r.

The algorithm concludes with a loop to repeatedly boost the exponents of the modal integers until they reach or exceed the values they have in r, while using gcd(•, r) to clip the values back to their values in r when they get oversize.

repeat {
$i_o \leftarrow i$

$i \leftarrow \gcd(\, i_o^2\,,\, r\,)$ // note, clipping against r now instead of against m.
} while ($i_o \neq i$) // repeat until i_o stops changing

i is now the modal integer factor of the original residue r.

If r is a pure modal integer then its modal unit factor is an euler but, in any case, the base unit factor u_o is obtained by euclidean division as

$$u_o \leftarrow r \;.\!\div\!.\; i$$

This "base" is only one of the number $n = r$ (int) alternative values the modal unit factor u of r (mod m) can assume. The complete set containing n values is expressed by

$$u \in u_o\Delta\delta \text{ where } \delta = m \;.\!\div\!.\; n \text{ and } n = |C_1| \;.\!\div\!.\; |C_r|$$

using the abbreviation $u_o\Delta\delta$ for the set $\{\, u_o + j \cdot \delta$ for $j = 0,1,2,\cdots,n-1 \,\}$ ◼

In my iOS app *Frrraction* this algorithm is implemented as { hp(MichiInt) } and is built into *Frrraction*'s division operation when *Frrraction* is in MPR-Residue Mode.

If *Frrraction*'s option switch is off then the / operation to divide numerator by denominator just shows the modular quotient, but if the switch is on then various properties of the denominator residue are also available: i-factor int_d, u-factor unit_d, inv_unit_d, omega_unit_d, all (c;e)-quotients, and all (i;u)-quotients.

Example (i;u) factorizations

r type	*r*	center of r	center of r^2	i = #of (i;u) q's	i_r	u_r
euler unit	43	1	1	1	1	
				1		43
mixed i·u	68	4	8	2	2	
				4	4	
				4		17
modal unit	56	8	8	1	1	
				1		56
mixed i·u	48	24	72	3	3	
				3		16
mixed i·u	54	18	36	2	2	
				2		27
modal unit	45	9	9	1	1	
				1		45