Instructions

shahargottlieb

January 2020

1 Instructions for enumerate_over_gcf.py

this program is mean to find hits of the type:

$$LHS = RHS$$

where $LHS = \frac{a \cdot Const + b}{c \cdot Const + d}$ and $RHS = a_0 + \frac{b_0}{a_1 + \frac{b1}{a_2 + \frac{b_2}{a_3 + \frac{b_4}{a_5 + \frac{b_5}{...}}}}}$

this idea is to create a hash table for all LHS possibilities, and then enumerate over the RHS and find hits. the search includes 3 steps:
1) initialize the LHS hash table.
2) first step enumeration - calculate RHS possibilities to finite low precision results, and find hits in hash table (this is the bottle neck).
3) verify the results from step 2, and print the true results.
the search is implemented also for multi-processing.
the parameters for the search are:

- **sym_constant**: the constant to search on (used in step 1 and 3) passd as a sympy constant.

- **lhs_search_limit** (uint): the limit for the hash tables parameters. all parameters for LHS (a,b,c,d) will be ranged from 0 to the limit, as well as the negatives of all permutations.

- **poly_a** (list of lists): $\{a_i\}_{i=0}^{\infty}$ is defined in default as a polynomial of the the form: $n(n(...(c_0 \cdot n + c_1) + c_2)...) + c_k$. poly_a are the options for c_k coefficients. step 2 will enumerate over all permutations. for example - if poly_a=[[1,2],[3,4]], this will enable 4 permutations for polynomials of degree 1: $a_n = \{n + 3, n + 4, 2n + 3, 2n + 4\}$

- **poly_b**: exactly the same as poly_a, with one change - the negatives of all coefficients will also be included in permutations. for the same example as before, the permutations will be: $b_n = \{n + 3, n + 4, 2n + 3, 2n + 4, -n - 3, -n - 4, -2n - 3, -2n - 4\}$ (8 permutations).

- **num_cores**: number of processes to use in search. the work will be divided between them by division of the first poly_a coefficient.

- **manual_splits_size** (optional): manual splitting of the work between the cores. for example, if poly_a = [range(12), range(12)] and we want to divide the work to 36 cores, we can use manual_splits_size = [2,2] to divide the work efficiently.

- **saved_hash**: path for existing hash table (in order to skip step 1). if path doesn't exist, a new hash table will be created and saved with this name.

- **create_an_series**: custom function for creation of a_n series (instead of polynomial). for example you can look at 'series_generators.create_series_zeta3_an')
- **create_bn_series**: same as a_n

2 examples for running enumerate_over_gcf.py

A lot of e results:

- **sym_constant** = sympy.E
- **lhs_search_limit** = 30
- **poly_a** = [[i for i in range(25)]] * 2
- **poly_b** = [[i for i in range(25)]] * 2
- **num_cores** = 1
- **manual_splits_size** = None
- **saved_hash** = os.path.join('hash_tables', 'e_30_hash.p')

zeta2 with our "hypothesis" on b_n:

- **sym_constant** = sympy.zeta(2)
- **lhs_search_limit** = 20
- **poly_a** = [[i for i in range(12)]] * 3
- **poly_b** = [[i for i in range(10)]] * 2
- **num_cores** = 4
- **manual_splits_size** = None
- **saved_hash** = os.path.join('hash_tables', 'zeta2_20_hash.p')
- **create_an_series** = None
- **create_bn_series** = partial(create_zeta_bn_series, 4)