T-Primes Generation Problem

Sampling

Currently, generating a sample of T-Primes in the range $(10^a, 10^b)$ biases strongly towards the larger t-primes. For the exponent's sub range $(b - d, b)$, where d is the difference in minimum and maximum exponents for the sample and $a \leq b - d \leq b$, we can compute the proportion of values that will fall in that range with

$$10^a \leq X \leq 10^b$$

$$P(X \geq 10^{b-d}) = \frac{10^b - 10^{b-d}}{10^b}$$

$$P(X \geq 10^{b-d}) = 1 - 10^{-d}$$

The larger the difference in exponents, the larger the asymmetry in a random sample. This can be attenuated to by dividing the exponent range (a, b) into j sub samples. This guarantees that each sub sample is approximately equally represented, only suffering from the asymmetry within each sub sample.

Future Frequency Updating

Keeping d small better distributes the values we are searching for. However, it also comes with the risk of an insufficient sample range to generate t-primes. T-primes, being defined as the square of a prime number, are scarce at the lower exponent ranges (e.g. 4 in the range $[10^1, 10^2)$). Should our grouping strategy request 10 cases from that example range and 50% of them t-primes, it would only be able to return 40% of the sample as t-primes. On the aggregate, this drags the frequency of positives down. To better represent this problem, we can use the definition of a PDF.

$$P(X = 1) = \sum_{k=1}^{j} P_k(X = 1)P(K = k)$$

Where j is the total number of groups, X is the expected outcome $(1, 0)$, and K is the frequency of that sample range in the overall sample. We assume this to be to uniform, and can thus be written as

$$P(K = k) = \frac{b - a}{j}$$

$$P(X = 1) = \frac{b - a}{j} \sum_{k=1}^{j} P_k(X = 1)$$

$$j\frac{P(X=1)}{b-a} = \sum_{k=1}^{j} P_k(X=1)$$

We can go further and bifurcate the sum by saying the last completed sample was sample $k = h$.

$$j\frac{P(X=1)}{b-a} = \sum_{k=1}^{h} P_k(X=1) + \sum_{k=h+1}^{j} P_k(X=1)$$

$$j\frac{P(X=1)}{b-a} - \sum_{k=1}^{h} P_k(X=1) = \sum_{k=h+1}^{j} P_k(X=1)$$

By dividing by $j - h$, we now have an expected value of the future samples' necessary $P_k(X = 1)$ to ensure an overall frequency of $P(X = 1)$. We can denote this new target frequency as μ_{h+1}

$$\mu_{h+1} = j\frac{P(X=1)}{(b-a)(j-h)} - \frac{1}{j-h}\sum_{k=1}^{h} P_k(X=1)$$

Converting to a more intuitive and iterable form, we can reduce some to constants

$$c_0 = j\frac{P(X=1)}{(b-a)}$$

$$\mu_{i+1} = \frac{c_0}{j-i} - \frac{1}{j-i}\sum_{k=1}^{i} P_k(X=1)$$

$$\mu_{i+1} = \frac{1}{j-i}(c_0 - \sum_{k=1}^{i} P_k(X=1))$$

Dynamic Frequency Updating Pseudocode

```python
def generate(a, b, r, n, balance=False, j=1):
        '''
        a: lower bound for exponent
        b: upper bound for exponent
        r: desired frequency of t-primes in sample
        n: size of sample
        balance: to create sub samples or not
        j: if balancing, number of sub samples
        '''
        if balance:
            c_0 = j*r/(b-a)
            r_i = r
            d = (b-a)/j
            nsub = nsub/j
            samples = set()

            for k in range(1, j):
                sample = generate(a=a+d*(k-1), b=a+d*k, r=r_0, n=nsub, balance=False)
                validate_sample()
                update_expected_frequency()
                samples.add(sample)
        else:
            generate_rn_t_primes()
        return samples
```