Supplementary Material for: Accurate small tail probabilities of sums of iid lattice-valued random variables via FFT

Huon Wilson and Uri Keich
School of Mathematics and Statistics, University of Sydney

September 26, 2016

7 Supplementary Figures and Tables

s_0	Saddlepoint	Normal	Exact
0	1.0E0	9.0E−1	1.0E0
1	8.5E−1	8.0E−1	8.4E−1
...			
10	1.8E−3	6.0E−5	5.0E−3
...			
200	4.6E−44	$*$	4.1E−43
201	2.8E−44	$*$	2.1E−43
202	NaN	$*$	9.8E−44
...			
206	2.2E−20	$*$	3.9E−45
...			
223	4.5E−49	$*$	3.8E−51
224	NaN	$*$	1.8E−51
...			
396	NaN	$*$	1.6E−86
397	∞	$*$	7.7E−87
398	∞	$*$	3.3E−87
399	NaN	$*$	1.1E−87
400	∞	$*$	2.2E−88

Table 1: The computation of $P = \sum_{s \geq s_0} \boldsymbol{p}^L(s)$ for a variety of s_0, via three methods: a saddlepoint approximation, a normal approximation and an exact method (NC). Here $L = 4$ and $\boldsymbol{p}(s) = A \cdot \exp\left[50(s/100 - 1)^2\right]$ for $s = 0, \ldots, 100$, where A guarantees $\boldsymbol{p}$ is a pmf, that is, $\|\boldsymbol{p}\|_1 = 1$. As expected, the saddlepoint approach offers a fairly good approximation of P on a considerably larger domain than the normal approach does (the $*$ entries are all less than 1E−2000). Still, in terms of relative error, it badly fails for almost all values $s_0 \geq 202$ and it does so inconsistently: underestimating P for some s_0 values, overestimating it for others, as well as suffering from numerical errors that produce non-usable results. For example, all s_0 from 224 to 396 give $P = \text{NaN}$ when computed via the saddlepoint approximation. We specifically implemented the saddlepoint method with the second continuity correction in (Butler, 2007, Section 1.2.3).

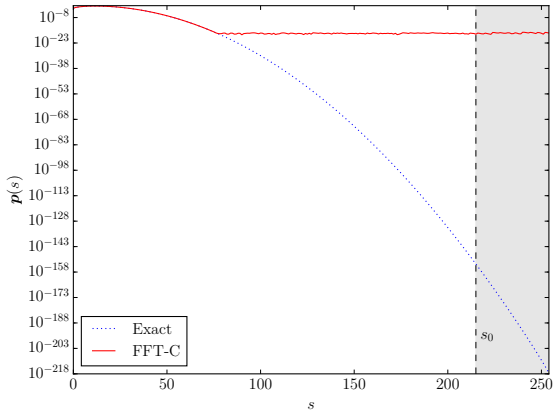

Figure 1: The values of the pmf $\boldsymbol{p}^{*2}$ (solid line) as computed via FFT-C per (1), are compared with the values computed via the accurate naive-convolution (dotted line). Here $\boldsymbol{p}(s) = A \cdot \exp\left(\frac{1}{60}s(10-s)\right)$ for $s = 0, 1, \ldots, 127$, where A guarantees $\boldsymbol{p}$ is a pmf. In this case, the p-value of $s_0 = 215$, which is the sum $P = \sum_{s \geq s_0} \boldsymbol{p}^{*L}(s)$ (the shaded region), is computed to be $\widetilde{P} \approx 3 \cdot 10^{-16}$ when the convolution is performed via FFT-C. However, the true value is $P \approx 6 \cdot 10^{-154}$, hence FFT's computed value is off by more than 138 orders of magnitude.

Note that the departure of the FFT-C computed pmf from the exact one is roughly at 10^{-16}, the machine precision, ε, of the `binary64` type used in this computation. The `binary64` type of the IEEE754-2008 (IEEE Computer Society, 2008) was called double precision in previous versions of that standard, and still has that name in some programming languages. It has $\varepsilon = 2^{-53} \approx 10^{-16}$. Note that the terms machine precision and ε are often referring to the distance from 1 to the next largest floating point number and this distance is exactly 2 times our ε.

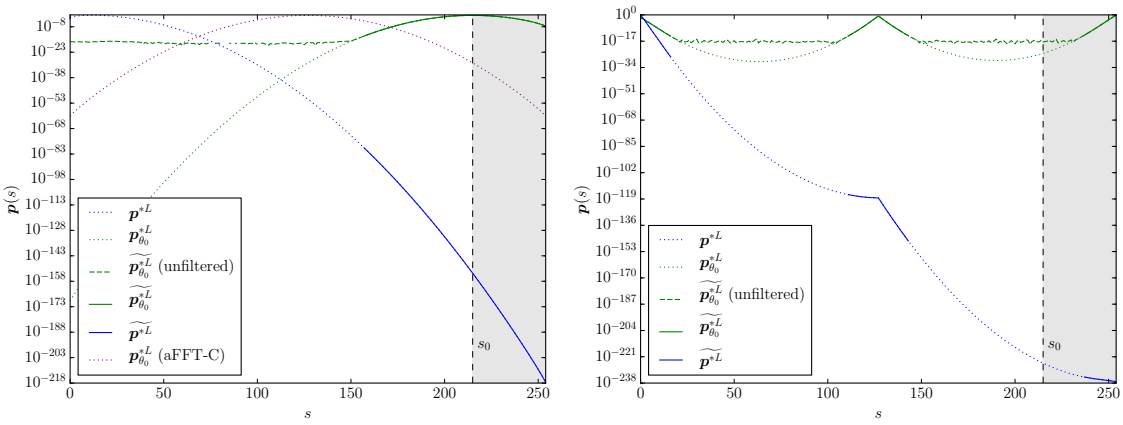

Figure 2: Illustrations of the sFFT algorithm computing an approximation $\widetilde{P}$ to $P = \sum_{s \geq s_0} \boldsymbol{p}^{*L}(s)$ for $s_0 = 215$ and $L = 2$ with two different pmfs $\boldsymbol{p}$. The first panel uses $\boldsymbol{p}$ from Supp. Figure 1, while the second panel studies $\boldsymbol{p}(s) = A \exp\left(\frac{1}{60} s(s-256)\right)$ for $s = 0, \ldots, 127$, with A such that $\|\boldsymbol{p}\|_1 = 1$. Both panels show the vectors computed at each stage of the algorithm: the dotted lines show exact values of the convolutions (computed via NC), solid lines show the values computed by sFFT after filtering errors out (zeroing values less than the error bound of Lemma 1), and the dashed green line shows the noise in the FFT-C computation that was removed. As can be seen, the solid blue line matches the exact value of $\boldsymbol{p} * \boldsymbol{p}$ very closely around s_0 in the first panel, and, indeed in that case, $\mathrm{rel}(\widetilde{P}, P) < 10^{-13}$. The second panel demonstrates how sFFT can fail with non-log-concave pmfs: the region around s_0, which contributes most to P, is still small in the shifted pmf, and hence it is not calculated accurately by FFT-C. This results in $\widetilde{P} \approx 2 \cdot 10^{-234}$ with the true value $P \approx 1 \cdot 10^{-225}$ many times larger. The dotted purple line in the first panel gives $\boldsymbol{p}^{*L}_{\theta_0}$ where θ_0 is computed for aFFT-C, designed to optimize the computation of the entire pmf $\boldsymbol{p}^{*L}$. Note that aFFT-C's choice reduces the range of values compared to the shift chosen by sFFT (dotted green curve), as well as the range of the unshifted pmf (blue curve). However, for computing P, it suffices to accurately recover the pmf only for indices greater than s_0, and on that region the sFFT chosen shift is producing a much smaller range of values. In particular, recovering the entire pmf via aFFT-C requires accurately computing a much larger range of values, placing an unnecessary burden of the FFT approach.

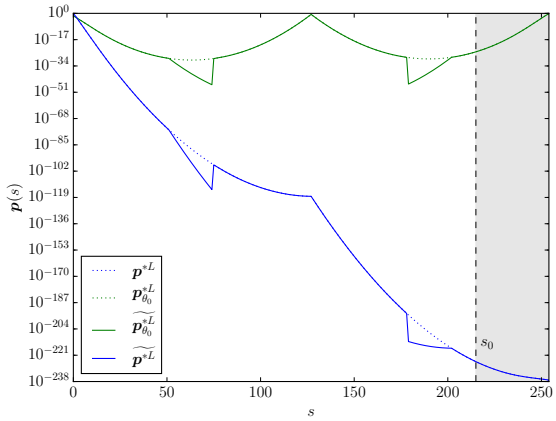

Figure 3: An illustration of the process of sisFFT's main pathway computing $P = \sum_{s \geq s_0} \boldsymbol{p}^{*L}(s)$ with $\boldsymbol{p}$, $L = 2$ and $s_0 = 215$ as in the second panel of Supp. Figure 2. The plot indicates that sisFFT is able to accurately compute the largest values of $\boldsymbol{p}^{*L}(s)$, indeed, it recovers all of them, thus giving an accurate estimate of P. The plot also hints at the use of the lower bound: the two jagged regions are the consequence of ignoring a few small values that are known to be unnecessary. Note that this plot is just for demonstrative purposes: since $L = 2$ in this example sisFFT would accurately compute the p-value with no explicit convolutions at all.

8 Bounds

8.1 Lower bound

A lower bound P_ℓ on $P = P(s_0)$ allows us to ignore values that are sufficiently smaller than P_ℓ, where the exact threshold is determined by the desired relative accuracy level. We begin with showing how to efficiently compute a non-trivial lower bound.

The main component of our lower bound computation is motivated by sFFT: the largest values of an iterated convolution can be computed efficiently and accurately via FFT-C. In general, however, just knowing the largest values will not give a tight lower bound on P: we saw an example of this in Supp. Figure 2, where sFFT computed an estimate, $\widetilde{P}$, many times smaller than the true one (second panel). This failure comes from the combination of FFT-C and the exponential shift: FFT-C can only accurately compute the larger values of the shifted pmf, $\boldsymbol{p}_{\theta_0}^{*L}$, leaving gaps that may become significant when the shift is reversed.

We can fill in some of those gaps by trivially noting that $\boldsymbol{p}_{\theta_0}^{*L} = \boldsymbol{p}_{\theta_0}^{*(L-1)} * \boldsymbol{p}_{\theta_0}$. Hence, rather than compute $\boldsymbol{p}_{\theta_0}^{*L}$ directly by FFT-C, we can compute the $(L-1)$-fold convolution, $\widetilde{\boldsymbol{p}_{\theta_0}^{*(L-1)}}$ in the manner of sFFT, and then convolve it with $\boldsymbol{p}_{\theta_0}$. This way, we utilize the largest values of $\boldsymbol{p}_{\theta_0}^{*(L-1)}$ combined with all the values of $\boldsymbol{p}_{\theta_0}$, thereby deriving a more effective bound than we can get using just the largest values of $\boldsymbol{p}_{\theta_0}^{*L}$ computed via FFT-C.

As detailed next, in practice, we do something equivalent to the above that instead computes $\boldsymbol{p}^{*(L-1)}$ with a variant of sFFT (using θ_0), and convolves the result with $\boldsymbol{p}$ using a variant of NC. A naive application of NC to compute the latter pairwise convolution would have a runtime complexity of $O(Ln^2)$, which, unfortunately, introduces a quadratic term. However, we are only interested in the tail sum, and as we will see next, this can be computed with a variant of NC in $O(Ln)$ time. Therefore, as computing $\widetilde{\boldsymbol{p}^{*(L-1)}}$ with sFFT takes only $O(Ln \log Ln)$ time, this is also the total time required to compute our lower bound for P.

While the optimal lower bound is typically attained using the exponential shift of θ_0, as determined by (5), it is beneficial to leave ourselves the option of using an arbitrary value of θ. Our procedure, which is summarized as Algorithm 1, starts with applying FFT-C (1) to $\boldsymbol{p}_\theta$ to compute $\widetilde{\boldsymbol{v}} := \widetilde{\boldsymbol{p}_\theta^{*(L-1)}}$ using a Q-dimensional DFT, where $Q = 2^K \geq N_{L-1}$ and $N_k = k(n-1)+1$ is the length of $\boldsymbol{p}^{*k}$.

We then proceed to filter out the roundoff error noise in $\widetilde{\boldsymbol{v}}$ by defining

$$\widetilde{\boldsymbol{v}}_\ell(k) := \begin{cases} \widetilde{\boldsymbol{v}}(k) - E & \text{if } \widetilde{\boldsymbol{v}}(k) > E \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where $E := (L-1) \cdot cK\varepsilon$ is the upper bound on the error per (2) of Lemma 1.

Note that $\widetilde{\boldsymbol{v}}_\ell$ is an estimate of $\boldsymbol{p}_\theta^{*(L-1)}$ that in turn allows us to estimate $\boldsymbol{p}^{*(L-1)}$ as

$$\widetilde{\boldsymbol{p}^{*(L-1)}}(k) := \widetilde{\boldsymbol{v}}_\ell(k) e^{-k\theta + (L-1)\kappa(\theta)}. \tag{9}$$

We next efficiently compute

$$\overline{\boldsymbol{p}}(k) := \sum_{j \geq k} \boldsymbol{p}(j), \tag{10}$$

by noting that $\overline{\boldsymbol{p}}(k) = \boldsymbol{p}(k) + \overline{\boldsymbol{p}}(k+1)$, and we use it to construct our lower bound of

$$\widetilde{P_\ell} := \sum_{k \geq 0} \widetilde{\boldsymbol{p}^{*(L-1)}}(k) \overline{\boldsymbol{p}}(s_0 - k), \tag{11}$$

where, for any vector $\boldsymbol{x}$, we take $\boldsymbol{x}(i) = 0$ when $i < 0$ or $i \geq \text{length}(\boldsymbol{x})$, which ensures the sums defining both $\overline{\boldsymbol{p}}$ and $\widetilde{P_\ell}$ are finite.

The correctness, that is $\widetilde{P_\ell} \leq P$, follows immediately from the following two claims.

Claim 4.

$$\widetilde{\boldsymbol{p}^{*(L-1)}}(k) \leq \boldsymbol{p}^{*(L-1)}(k).$$

Before proving the claim we note that the LHS is considered to be precisely computed except for the FFT-C induced errors, that is, the roundoff errors introduced in (9) are ignored. This is justified since these errors are negligible compared with the errors introduced by FFT-C. It is straightforward to incorporate those negligible errors into the calculation but we felt it adds an unnecessary complication.

Proof. By Lemma 1, we have

$$\left\| \widetilde{\boldsymbol{v}} - \boldsymbol{p}_\theta^{*(L-1)} \right\|_\infty \le E,$$

and hence $\widetilde{\boldsymbol{v}_\ell}(k) \le \boldsymbol{p}_\theta^{*(L-1)}(k)$ for all k. Multiplying both sides by $e^{-k\theta+(L-1)\kappa(\theta)}$ proves the claim. □

Claim 5.

$$P = \sum_{k \ge s_0} \boldsymbol{p}^{*L}(k) = \sum_{k \ge 0} \boldsymbol{p}^{*(L-1)}(k)\overline{\boldsymbol{p}}(s_0 - k).$$

Proof. Let $\boldsymbol{q}$ be a pmf of the $\mathbb{N}_0$-valued RV Y, which is independent of the $\mathbb{N}_0$-valued RV X with pmf $\boldsymbol{p}$. Then

$$P(X + Y \ge s_0) = \sum_{k \ge 0} P(Y = k)P(X \ge s_0 - k) = \sum_{k \ge 0} \boldsymbol{q}(k)\overline{\boldsymbol{p}}(s_0 - k).$$ □

Since all vectors are non-negative it follows immediately that

$$\widetilde{P_\ell} = \sum_{k \ge 0} \widetilde{\boldsymbol{p}^{*(L-1)}}(k)\overline{\boldsymbol{p}}(s_0 - k) \le \sum_{k \ge 0} \boldsymbol{p}^{*(L-1)}(k)\overline{\boldsymbol{p}}(s_0 - k) = P. \tag{12}$$

Again, the comment after Supp. Claim 4 about ignoring negligible roundoff errors applies here in computing $\overline{\boldsymbol{p}}$ in (10), and $\widetilde{P_\ell}$ in (11).

8.2 Upper bound

It is not difficult to see that in cases where the previously published sFFT is deemed sufficiently accurate, our lower bound P_ℓ will also not deviate significantly from P. In such cases we can save a significant amount of computation by avoiding the main pathway of our new algorithm.

Below we introduce P_u, an upper bound analogous to P_ℓ. Combined, the two bounds allow us to efficiently identify those cases where we can "short-circuit" our main algorithm because the bounds are already tight enough. In such cases, we interpolate the bounds to produce an estimate $\widetilde{P}$ that satisfies the required accuracy.

Define

$$\widetilde{\boldsymbol{v}_u}(k) := \widetilde{\boldsymbol{v}}(k) + E, \tag{13}$$

where E is the same bound on the error in computing $\widetilde{\boldsymbol{v}} := \widetilde{\boldsymbol{p}_\theta^{*(L-1)}}$ using FFT that we defined just after (8).

Note that $\widetilde{\boldsymbol{v}_u}(k) \ge \boldsymbol{p}_\theta^{*(L-1)}(k)$ for all k, and therefore with

$$\widetilde{\boldsymbol{q}^{*(L-1)}}(k) := \widetilde{\boldsymbol{v}_u}(k)e^{-k\theta+(L-1)\kappa(\theta)}, \tag{14}$$

similarly to Supp. Claim 4 we have

$$\widetilde{\boldsymbol{q}^{*(L-1)}}(k) \ge \boldsymbol{p}^{*(L-1)}(k).$$

Thus, we can define the upper bound

$$\widetilde{P_u} := \sum_{k \ge 0} \widetilde{\boldsymbol{q}^{*(L-1)}}(k)\overline{\boldsymbol{p}}(s_0 - k), \tag{15}$$

and, again from Supp. Claim 5, we indeed have

$$\widetilde{P_\ell} \le P \le \widetilde{P_u}. \tag{16}$$

8.3 Complexity of computing the bounds

As mentioned, the runtime complexity of Algorithm 1 is $O(Ln \log Ln)$, which can be seen as follows:

- Computing $\widetilde{\boldsymbol{v}}$ takes $O(Ln \log Ln)$, followed by $O(Ln)$ steps to compute $\widetilde{\boldsymbol{v}_\ell}$, $\widetilde{\boldsymbol{v}_u}$, $\widetilde{\boldsymbol{p}^{*(L-1)}}$, and $\widetilde{\boldsymbol{q}^{*(L-1)}}$.
- Using the relation $\overline{\boldsymbol{p}}(k) = \boldsymbol{p}(k) + \overline{\boldsymbol{p}}(k+1)$, the vector $\overline{\boldsymbol{p}}$ can be computed in $O(n)$.
- Finally, computing $\widetilde{P_\ell}$ through (11) and $\widetilde{P_u}$ via (15) takes another $O(N_{L-1} - s_0) = O(Ln)$ steps.

9 Error analysis of sisFFT's main pathway

We showed that aFFT-C controls the relative error in convolving the pair of non-negative vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ (Wilson and Keich, 2016). That is, with aFFT-C$(\boldsymbol{v}, \boldsymbol{w}, \alpha)$ denoting $\widetilde{\boldsymbol{v} * \boldsymbol{w}}$ computed using aFFT-C with accuracy parameter $\alpha \geq 2$,

$$\left(1 - \frac{1}{\alpha}\right)(\boldsymbol{v} * \boldsymbol{w})(k) \leq \text{aFFT-C}(\boldsymbol{v}, \boldsymbol{w}, \alpha)(k) \leq \left(1 + \frac{1}{\alpha}\right)(\boldsymbol{v} * \boldsymbol{w})(k), \tag{17}$$

for all k.

Let

$$C_{\alpha,\Delta}(\boldsymbol{v}, \boldsymbol{w}) = \text{trim-aFFT-C}(\boldsymbol{v}, \boldsymbol{w}, \alpha, \Delta),$$

where the RHS denotes the application of trim-aFFT-C (Algorithm 3), with accuracy parameter $\alpha \geq 2$ and trim level $\Delta \geq 0$, to the non-negative vectors $\boldsymbol{v}$ and $\boldsymbol{w}$. It follows from (17) and non-negativity that

$$\left(1 - \frac{1}{\alpha}\right)(\boldsymbol{v}_{\geq\Delta} * \boldsymbol{w}_{\geq\Delta})(k) \leq C_{\alpha,\Delta}(\boldsymbol{v}, \boldsymbol{w})(k) \leq \left(1 + \frac{1}{\alpha}\right)(\boldsymbol{v} * \boldsymbol{w})(k), \tag{18}$$

for all k.

The following theorem provides bounds on the accumulation of errors in computing the convolution $\widetilde{\boldsymbol{q}^{*L}}$ using Squaring-C implemented with trim-aFFT-C. Note that the theorem applies just as well to any pairwise convolution algorithm that satisfies (18). Our assumption on the convolved input pmf $\boldsymbol{q}$ is that it can be perfectly represented as an n-dimensional floating-point vector, that is, $\widetilde{\boldsymbol{q}} = \boldsymbol{q}$.

Theorem 6. *Suppose $\boldsymbol{q}$ is a pmf such that $\widetilde{\boldsymbol{q}} = \boldsymbol{q}$. Let $L \geq 2$ be the convolution order, and let its binary representation be $L = \sum_i b_i 2^i$. Let $\ell = \sum_i b_i$, and let $0 < i_1 < \ldots < i_\ell = \lfloor \log_2 L \rfloor$ be the indices i such that $b_i = 1$. Let $\alpha \geq 2$ be the accuracy parameter, and let $\Delta \in [0, 1)$ be the trimming parameter of $C_{\alpha,\Delta}$ (trim-aFFT-C with accuracy α and trimming Δ). Finally, with $\beta = 1/\alpha$, the accuracy level, define*

$$L_j = \sum_{k=1}^{j} b_{i_k} 2^{i_k} \tag{19}$$

$$r_i = (1+\beta)^{2^i - 1} - 1 \tag{20}$$

$$\bar{r}_j = (1+\beta)^{L_j - 1} - 1 \tag{21}$$

$$\delta_i = \Delta \sum_{k=0}^{i-1} 2^{i-k}(1 + r_k) \tag{22}$$

$$\bar{\delta}_j = \sum_{k=1}^{j} \delta_{i_k} + \Delta \sum_{k=1}^{j-1} \left(2 + \bar{r}_k + r_{i_{k+1}}\right). \tag{23}$$

If $\widetilde{\boldsymbol{q}^{*L}}$ is computed via Squaring-C with $C(\alpha, \Delta)$, then

$$-\bar{r}_\ell \boldsymbol{q}^{*L}(k) - \bar{\delta}_\ell \leq \widetilde{\boldsymbol{q}^{*L}}(k) - \boldsymbol{q}^{*L}(k) \leq \bar{r}_\ell \boldsymbol{q}^{*L}(k) \tag{24}$$

for all k.

Corollary 7. *Given a pmf $\boldsymbol{q}$ with $\widetilde{\boldsymbol{q}} = \boldsymbol{q}$, a convolution order $L \geq 2$ with the above binary expansion, an accuracy parameter $\alpha_0 \geq 2$ and a lower bound $\Delta_0 \in [0, 1)$, choose*

$$\alpha \geq \left[\left(1 + \frac{1}{\alpha_0} \right)^{1/(L-1)} - 1 \right]^{-1}, \tag{25}$$

and with $\beta = 1/\alpha$, and r_j and $\bar{r}_k$ defined as in (20) and (21) above, let

$$M = \sum_{k=1}^{\ell} \sum_{j=0}^{i_k - 1} 2^{i_k - j}(1 + r_j) + \sum_{k=1}^{\ell-1} \left(2 + \bar{r}_k + r_{i_{k+1}} \right). \tag{26}$$

Finally, choose

$$\Delta \leq \frac{\Delta_0}{M}. \tag{27}$$

Then, with $\widetilde{\boldsymbol{q}^{*L}}$ computed using Squaring-C with $C_{\alpha,\Delta}$ we have

$$-\frac{1}{\alpha_0} \boldsymbol{q}^{*L}(k) - \Delta_0 \leq \widetilde{\boldsymbol{q}^{*L}}(k) - \boldsymbol{q}^{*L}(k) \leq \frac{1}{\alpha_0} \boldsymbol{q}^{*L}(k) \tag{28}$$

for each k.

Again, while explicitly using trim-aFFT-C, the corollary holds for any pairwise convolution that satisfies (18).

Proof of Supplementary Corollary 7. Note that $L_\ell = L$ and hence by (21) and (25)

$$\bar{r}_\ell = (1 + \beta)^{L-1} - 1 = (1 + 1/\alpha)^{L-1} - 1 \leq \left[\left(1 + \frac{1}{\alpha_0} \right)^{1/(L-1)} \right]^{L-1} - 1 = \frac{1}{\alpha_0}.$$

Similarly, using (23), (22), and (26)

$$\begin{aligned}
\bar{\delta}_\ell &= \sum_{k=1}^{\ell} \delta_{i_k} + \Delta \sum_{k=1}^{\ell-1} \left(2 + \bar{r}_k + r_{i_{k+1}} \right) \\
&= \sum_{k=1}^{\ell} \left[\Delta \sum_{j=0}^{i_k - 1} 2^{i_k - j}(1 + r_j) \right] + \Delta \sum_{k=1}^{\ell-1} \left(2 + \bar{r}_k + r_{i_{k+1}} \right) \\
&= \Delta M \\
&\leq \Delta_0.
\end{aligned}$$

Taken together, (28) now follows trivially from (24) of Supp. Theorem 6. □

Proof of Supplementary Theorem 6. The proof progresses through three main steps.

The first step quantifies the cumulative error that $C(\alpha, \Delta)$ introduces if the input pmfs themselves have already been computed using likewise approximation:

Lemma 8. *Suppose $\boldsymbol{v}$ and $\boldsymbol{w}$ are pmfs and $\widetilde{\boldsymbol{v}}$ and $\widetilde{\boldsymbol{w}}$ are non-negative approximations such that, for each k,*

$$\begin{aligned}
(1 - e_1)\boldsymbol{v}(k) - \varepsilon_1 \leq \widetilde{\boldsymbol{v}}(k) \leq (1 + e_1)\boldsymbol{v}(k) \\
(1 - e_2)\boldsymbol{w}(k) - \varepsilon_2 \leq \widetilde{\boldsymbol{w}}(k) \leq (1 + e_2)\boldsymbol{w}(k),
\end{aligned} \tag{29}$$

where $e_i, \varepsilon_i \in [0, 1)$. Then, with $\beta = 1/\alpha$,

$$C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{w}})(k) \leq (1 + \beta)(1 + e_1)(1 + e_2)(\boldsymbol{v} * \boldsymbol{w})(k) \tag{30}$$

$$C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{w}})(k) \geq (1 - \beta)(1 - e_1)(1 - e_2)(\boldsymbol{v} * \boldsymbol{w})(k) - (\varepsilon_1 + \varepsilon_2 + \Delta(2 + e_1 + e_2)) \tag{31}$$

for all k.

Proof of Supplementary Lemma 8. To prove the lemma we need the following claim, which bounds the difference between the exactly computed $\boldsymbol{v}_{\geq\Delta} * \boldsymbol{w}_{\geq\Delta}$ and $\boldsymbol{v} * \boldsymbol{w}$.

Claim 9. *Given non-negative vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, and a lower bound $\Delta \geq 0$, we have*

$$0 \leq (\boldsymbol{v} * \boldsymbol{w})(k) - (\boldsymbol{v}_{\geq\Delta} * \boldsymbol{w}_{\geq\Delta})(k) \leq \Delta(\|\boldsymbol{v}\|_1 + \|\boldsymbol{w}\|_1) \tag{32}$$

for all k.

Proof of Supplementary Claim 9. Since the vectors are non-negative and no element of the vectors $\boldsymbol{v}_{\geq\Delta}$ and $\boldsymbol{w}_{\geq\Delta}$ is larger than the corresponding element of $\boldsymbol{v}$ and $\boldsymbol{w}$ respectively, the left hand inequality of (32) is clear.

Let $\boldsymbol{v}_{<\Delta} = \boldsymbol{v} - \boldsymbol{v}_{\geq\Delta}$ (and similarly for $\boldsymbol{w}_{<\Delta}$). Then

$$\begin{aligned}
\|\boldsymbol{v} * \boldsymbol{w} - \boldsymbol{v}_{\geq\Delta} * \boldsymbol{w}_{\geq\Delta}\|_\infty &= \|\boldsymbol{v}_{<\Delta} * \boldsymbol{w}_{\geq\Delta} + \boldsymbol{v}_{\geq\Delta} * \boldsymbol{w}_{<\Delta} + \boldsymbol{v}_{<\Delta} * \boldsymbol{w}_{<\Delta}\|_\infty \\
&= \|\boldsymbol{v}_{<\Delta} * \boldsymbol{w} + \boldsymbol{v}_{\geq\Delta} * \boldsymbol{w}_{<\Delta}\|_\infty \\
&\leq \|\boldsymbol{v}_{<\Delta}\|_\infty \|\boldsymbol{w}\|_1 + \|\boldsymbol{v}_{\geq\Delta}\|_1 \|\boldsymbol{w}_{<\Delta}\|_\infty \\
&\leq \Delta(\|\boldsymbol{v}\|_1 + \|\boldsymbol{w}\|_1),
\end{aligned}$$

where we used $\|\boldsymbol{x} * \boldsymbol{y}\|_\infty \leq \|\boldsymbol{x}\|_\infty \|\boldsymbol{y}\|_1$. This proves the right hand side of (32). □

Returning to the proof of the lemma, since the vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ are non-negative, we have

$$C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{w}})(k) \leq (1+\beta)(\widetilde{\boldsymbol{v}} * \widetilde{\boldsymbol{w}})(k) \leq (1+\beta)(1+e_1)(1+e_2)(\boldsymbol{v} * \boldsymbol{w})(k)$$

for each k, proving (30).

As for the other inequality, define the vectors $\boldsymbol{\varepsilon}'_1$ and $\boldsymbol{\varepsilon}'_2$ as

$$\begin{aligned}
\boldsymbol{\varepsilon}'_1(k) &= \min((1-e_1)\boldsymbol{v}(k), \varepsilon_1) \\
\boldsymbol{\varepsilon}'_2(k) &= \min((1-e_2)\boldsymbol{w}(k), \varepsilon_2).
\end{aligned}$$

Since $\widetilde{\boldsymbol{v}}$ is non-negative, we have $\widetilde{\boldsymbol{v}}(k) \geq (1-e_1)\boldsymbol{v}(k) - \boldsymbol{\varepsilon}'_1(k)$ and similarly for $\widetilde{\boldsymbol{w}}$. Therefore,

$$\begin{aligned}
(\widetilde{\boldsymbol{v}} * \widetilde{\boldsymbol{w}})(k) &\geq (1-e_1)(1-e_2)(\boldsymbol{v} * \boldsymbol{w})(k) - (1-e_1)(\boldsymbol{v} * \boldsymbol{\varepsilon}'_2)(k) - (1-e_2)(\boldsymbol{\varepsilon}'_1 * \boldsymbol{w})(k) + (\boldsymbol{\varepsilon}'_1 * \boldsymbol{\varepsilon}'_2)(k) \\
&\geq (1-e_1)(1-e_2)(\boldsymbol{v} * \boldsymbol{w})(k) - (\boldsymbol{v} * \boldsymbol{\varepsilon}'_2)(k) - (\boldsymbol{\varepsilon}'_1 * \boldsymbol{w})(k).
\end{aligned} \tag{33}$$

Looking at the subtracted terms, we can bound them from above:

$$\begin{aligned}
(\boldsymbol{v} * \boldsymbol{\varepsilon}'_2)(k) &\leq \|\boldsymbol{v}\|_1 \|\boldsymbol{\varepsilon}'_2\|_\infty \leq 1 \cdot \varepsilon_2 \\
(\boldsymbol{\varepsilon}'_1 * \boldsymbol{w})(k) &\leq \|\boldsymbol{\varepsilon}'_1\|_\infty \|\boldsymbol{w}\|_1 \leq \varepsilon_1 \cdot 1
\end{aligned}$$

By (18),

$$C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{w}})(k) \geq (1-\beta)(\widetilde{\boldsymbol{v}}_{\geq\Delta} * \widetilde{\boldsymbol{w}}_{\geq\Delta})(k) = (1-\beta)(\widetilde{\boldsymbol{v}} * \widetilde{\boldsymbol{w}} + \underbrace{\widetilde{\boldsymbol{v}}_{\geq\Delta} * \widetilde{\boldsymbol{w}}_{\geq\Delta} - \widetilde{\boldsymbol{v}} * \widetilde{\boldsymbol{w}}}_{\boldsymbol{\gamma}(k)})(k) \tag{34}$$

The term $\boldsymbol{\gamma}(k)$ can be bounded via Claim 9, which implies that

$$\begin{aligned}
\boldsymbol{\gamma}(k) = (\widetilde{\boldsymbol{v}}_{\geq\Delta} * \widetilde{\boldsymbol{w}}_{\geq\Delta} - \widetilde{\boldsymbol{v}} * \widetilde{\boldsymbol{w}})(k) &\geq -\Delta(\|\widetilde{\boldsymbol{v}}\|_1 + \|\widetilde{\boldsymbol{w}}\|_1) \\
&\geq -\Delta\left[(1+e_1)\|\boldsymbol{v}\|_1 + (1+e_2)\|\boldsymbol{w}\|_1\right] \\
&= -\Delta(2+e_1+e_2).
\end{aligned}$$

Bringing together this and (33) allows us to complete the reasoning of (34),

$$\begin{aligned}
C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{w}})(k) &\geq (1-\beta)\left[(\widetilde{\boldsymbol{v}} * \widetilde{\boldsymbol{w}})(k) - \Delta(2+e_1+e_2)\right] \\
&\geq (1-\beta)\left[(1-e_1)(1-e_2)(\boldsymbol{v} * \boldsymbol{w})(k) - (\varepsilon_1 + \varepsilon_2 + \Delta(2+e_1+e_2))\right] \\
&\geq (1-\beta)(1-e_1)(1-e_2)(\boldsymbol{v} * \boldsymbol{w})(k) - \left[\varepsilon_1 + \varepsilon_2 + \Delta(2+e_1+e_2)\right],
\end{aligned}$$

which is exactly (31). □

These bounds are almost in the form that allows iterated use of the result; the only problem is that the lower and upper bound on $C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}},\widetilde{\boldsymbol{w}})(k)$ do not have matching coefficients. However, it is easy to see that

$$(1-\beta)(1-e_1)(1-e_2)-1 \geq 1-(1+\beta)(1+e_1)(1+e_2),$$

giving the following result.

Corollary 10. *Suppose $\boldsymbol{v}$ and $\boldsymbol{w}$ are pmfs with approximations $\widetilde{\boldsymbol{v}}$ and $\widetilde{\boldsymbol{w}}$ satisfying* (29) *as in Supp. Lemma 8. Then with*

$$\begin{aligned} e_3 &= (1+\beta)(1+e_1)(1+e_2)-1 \\ \varepsilon_3 &= \varepsilon_1+\varepsilon_2+\Delta(2+e_1+e_2) \end{aligned} \tag{35}$$

for all k, we have

$$(1-e_3)(\boldsymbol{v}*\boldsymbol{w})(k)-\varepsilon_3 \leq C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}},\widetilde{\boldsymbol{w}})(k) \leq (1+e_3)(\boldsymbol{v}*\boldsymbol{w})(k). \tag{36}$$

Note that e_3 and ε_3 quantify the quality of $C_{\alpha,\Delta}(\widetilde{\boldsymbol{v}},\widetilde{\boldsymbol{w}})$ as an approximation to $\boldsymbol{v}*\boldsymbol{w}$ and that they are in the form, (29), required to use as an input to this same corollary. This allows us to iteratively use the corollary to bound the error in computing $\boldsymbol{q}^{*2^i}$, which bring us to the second main step of our proof.

As in Squaring-C (Algorithm 2), for the vector $\boldsymbol{q}$ define the sequence of convolutions

$$\widetilde{\boldsymbol{x}_0}=\widetilde{\boldsymbol{q}}=\boldsymbol{q} \qquad \widetilde{\boldsymbol{x}_i}=C_{\alpha,\Delta}(\widetilde{\boldsymbol{x}_{i-1}},\widetilde{\boldsymbol{x}_{i-1}}). \tag{37}$$

The $\widetilde{\boldsymbol{x}_i}$ are approximations to $\boldsymbol{x}_i=\boldsymbol{q}^{*2^i}$ and the following lemma quantifies their quality:

Lemma 11. *Let r_i be as in* (20)*, and let δ_i be as in* (22)*, then, for each k,*

$$-r_i\boldsymbol{x}_i(k)-\delta_i \leq \widetilde{\boldsymbol{x}_i}(k)-\boldsymbol{x}_i(k) \leq r_i\boldsymbol{x}_i(k) \tag{38}$$

Proof. We prove the result by induction on i. The relation (38) is clearly true for $i=0$ since we assumed $\boldsymbol{q}=\widetilde{\boldsymbol{q}}$.

Suppose the result is true for some $i\geq 0$ and let $\boldsymbol{v}=\boldsymbol{w}=\boldsymbol{x}_i$ (and so $\widetilde{\boldsymbol{v}}=\widetilde{\boldsymbol{w}}=\widetilde{\boldsymbol{x}}_i$). By the inductive hypothesis, $\widetilde{\boldsymbol{v}}$ and $\widetilde{\boldsymbol{w}}$ satisfy (29) with the accuracy coefficients $e_1=e_2=r_i$ and the lower bounds of $\varepsilon_1=\varepsilon_2=\delta_i$. Since $\boldsymbol{v}*\boldsymbol{w}=\boldsymbol{x}_{i+1}$ it follows from (36) that with e_3 and ε_3 defined in (35)

$$-e_3\boldsymbol{x}_{i+1}(k)-\varepsilon_3 \leq \widetilde{\boldsymbol{x}_{i+1}}(k)-\boldsymbol{x}_{i+1}(k) \leq e_3\boldsymbol{x}_{i+1}(k).$$

The induction is completed by recalling (20) to note that

$$e_3=(1+\beta)(1+r_i)^2-1=(1+\beta)\left[(1+\beta)^{2^i-1}\right]^2-1=r_{i+1},$$

and similarly, from (22) we have

$$\begin{aligned} \varepsilon_3 &= \delta_i+\delta_i+\Delta(2+r_i+r_i) \\ &= 2\left[\Delta\sum_{k=0}^{i-1}2^{i-k}(1+r_k)\right]+\Delta\cdot 2(1+r_i) \\ &= \Delta\sum_{k=0}^{i}2^{i+1-k}(1+r_k) \\ &= \delta_{i+1}. \end{aligned}$$

□

Finally, in the third main step of our proof we combine our results from the first two steps to prove the bounds on the accumulated errors.

Recall the binary expansion $L=\sum_i b_i 2^i$: we denoted the ℓ indices i for which $b_i=1$ by $i_1<i_2<\ldots<i_\ell$ and we defined $L_k=\sum_{j=1}^{k}b_{i_j}2^{i_j}$. Using these and the approximations $\widetilde{\boldsymbol{x}_i}$ to $\boldsymbol{x}_i=\boldsymbol{q}^{*2^i}$ defined in (37) we define the sequence:

$$\widetilde{\boldsymbol{v}_1}=\widetilde{\boldsymbol{x}_{i_1}} \qquad \widetilde{\boldsymbol{v}_j}=C_{\alpha,\Delta}(\widetilde{\boldsymbol{x}_{i_j}},\widetilde{\boldsymbol{v}_{j-1}}). \tag{39}$$

Clearly, each $\widetilde{\boldsymbol{v}_j}$ is the approximation to $\boldsymbol{v}_j=\boldsymbol{q}^{*L_j}$, computed iteratively by Squaring-C using $C(\alpha,\Delta)$. Therefore, the approximation $\widetilde{\boldsymbol{q}^{*L}}$ which is the focus of Supp. Theorem 6 is exactly $\widetilde{\boldsymbol{v}_\ell}$.

Lemma 12. *Let $\bar{r}_j$ be as defined in* (21) *and let $\bar{\delta}_j$ as in* (23)*, then, for each k,*

$$-\bar{r}_j \boldsymbol{v}_j(k) - \bar{\delta}_j \le \widetilde{\boldsymbol{v}_j}(k) - \boldsymbol{v}_j(k) \le \bar{r}_j \boldsymbol{v}_j(k) \tag{40}$$

Proof. This follows by induction on j using Supp. Corollary 10, in a similar manner to the proof of Supp. Lemma 11.

For $j = 1$ we have $\boldsymbol{v}_1 = \boldsymbol{x}_{i_1}$ by (39). At the same time, since $2^{i_1} = L_1$, it follows that from (21) that

$$\bar{r}_1 = (1+\beta)^{2^{i_1}-1} = r_{i_1},$$

and similarly from (23), that $\bar{\delta}_1 = \delta_{i_1}$. Thus, Supp. Lemma 11 implies that (40) holds for $j = 1$.

Suppose the result is true for some $j \ge 0$ and let $\boldsymbol{v} = \boldsymbol{v}_j$ and $\boldsymbol{w} = \boldsymbol{x}_{j+1}$. By the inductive hypothesis, $\widetilde{\boldsymbol{v}}$ satisfies (29) with the accuracy coefficient $e_1 = \bar{r}_j$ and the lower bound of $\varepsilon_1 = \bar{\delta}_j$. Using Supp. Lemma 11 again, we find that $\widetilde{\boldsymbol{w}}$ satisfies (29) with the accuracy coefficient $e_2 = r_i$ and the lower bound of $\varepsilon_2 = \delta_i$. Since $\boldsymbol{v} * \boldsymbol{w} = \boldsymbol{v}_{j+1}$, it follows from (36) of Supp. Corollary 10 that with e_3 and ε_3 defined in (35),

$$-e_3 \boldsymbol{v}_{j+1}(k) - \varepsilon_3 \le \widetilde{\boldsymbol{v}_{j+1}}(k) - \boldsymbol{v}_{j+1}(k) \le e_3 \boldsymbol{v}_{j+1}(k).$$

The inductive step is completed by noting that

$$e_3 = (1+\beta)(1+\bar{r}_j)(1+r_{i_{j+1}}) - 1 = (1+\beta)(1+\beta)^{L_j-1}(1+\beta)^{2^{i_{j+1}}-1} - 1 = (1+\beta)^{L_j+2^{i_{j+1}}-1} - 1 = \bar{r}_{j+1},$$

and similarly that

$$\begin{aligned}
\varepsilon_3 &= \bar{\delta}_j + \delta_{i_{j+1}} + \Delta(2 + \bar{r}_j + r_{i_{j+1}}) \\
&= \left[\sum_{k=1}^{j} \delta_{i_k} + \Delta \sum_{k=1}^{j-1} \left(2 + \bar{r}_k + r_{i_{k+1}}\right)\right] + \delta_{i_{j+1}} + \Delta(2 + \bar{r}_j + r_{i_{j+1}}) \\
&= \sum_{k=1}^{j+1} \delta_{i_k} + \Delta \sum_{k=1}^{(j+1)-1} \left(2 + \bar{r}_k + r_{i_{k+1}}\right) \\
&= \bar{\delta}_{j+1}.
\end{aligned}$$

□

This last lemma is precisely what is needed to complete the proof of the theorem: set $j = \ell$ in Supp. Lemma 12, then $\widetilde{\boldsymbol{v}}_\ell = \widetilde{\boldsymbol{q}^{*L}}$ and so (24) is exactly (40). □

Theorem 13. *If $\boldsymbol{p} = \widetilde{\boldsymbol{p}}$ is a pmf of length n, then with $P = \sum_{s \ge s_0} \boldsymbol{p}^{*L}(s)$ and $\widetilde{P}$ computed by sisFFT with accuracy parameter $\gamma \in (0, 1/2]$, we have*

$$\mathrm{rel}(\widetilde{P}, P) \le \gamma.$$

Note that the comment after Supp. Claim 4 about ignoring negligible roundoff errors in computations that involve only positive terms applies here as well.

Proof. The exact tail probability, P, can be written in terms of the shifted pmf $\boldsymbol{p}_{\theta_0}$ via (4):

$$P = \sum_{s \ge s_0} \boldsymbol{p}_{\theta_0}^{*L}(s) \cdot e^{-s\theta_0 + L\kappa(\theta_0)},$$

and hence the difference $\widetilde{P} - P$ can be expressed as

$$\widetilde{P} - P = \sum_{s \ge s_0} \left[\widetilde{\boldsymbol{p}_{\theta_0}^{*L}}(s) - \boldsymbol{p}_{\theta_0}^{*L}(s)\right] e^{-s\theta_0 + L\kappa(\theta_0)}.$$

Therefore, with our choice of $\alpha_0 = 2/\gamma$ and $\Delta_0 = B\gamma/2$, applying Supp. Corollary 7 to $\boldsymbol{q} = \boldsymbol{p}_{\theta_0}$, we have, for all k

$$\left(1 - \frac{\gamma}{2}\right) \boldsymbol{p}_{\theta_0}^{*L}(k) - \frac{B\gamma}{2} \le \widetilde{\boldsymbol{p}_{\theta_0}^{*L}}(k) \le \left(1 + \frac{\gamma}{2}\right) \boldsymbol{p}_{\theta_0}^{*L}(k). \tag{41}$$

It follows that

$$\widetilde{P} - P \leq \sum_{s \geq s_0} \frac{\gamma}{2} \cdot \boldsymbol{p}_{\theta_0}^{*L}(s) \cdot e^{-s\theta_0 + L\kappa(\theta_0)} = \frac{\gamma}{2} P. \tag{42}$$

At the same time, using the left hand inequality of (41), along with (12), and the definition of B in line 10 of Algorithm 4 we have

$$\begin{aligned} \widetilde{P} - P &\geq \sum_{s \geq s_0} \left[-\frac{\gamma}{2} \cdot \boldsymbol{p}_{\theta_0}^{*L}(s) - \frac{\gamma}{2} B \right] e^{-s\theta_0 + L\kappa(\theta_0)} \\ &= -\frac{\gamma}{2} \left[P + B \sum_{s=s_0}^{N_L - 1} e^{-s\theta_0 + L\kappa(\theta_0)} \right] \\ &= -\frac{\gamma}{2} \left(P + \widetilde{P_\ell} \right) \\ &\geq -\gamma P. \end{aligned} \tag{43}$$

Together, the last two inequalities, (42) and (43), establish $\text{rel}(\widetilde{P}, P) \leq \gamma$. □

10 Supplementary Empirical Results

We generated a variety of tail sums $\sum_{s \geq s_0} \boldsymbol{p}^L(s)$ with various values of s_0, $\boldsymbol{p}$ and L. The pmfs $\boldsymbol{p}$ were taken from the same classes as Wilson and Keich (2016) used to test aFFT-C: fix an integer n, and suppose each pmf $\boldsymbol{p}^{(k)}$ is a vector of length n. The values $a^{(k)}, b^{(k)}, c^{(k)} \sim U(0,1)$ are independent, and fixed for each $\boldsymbol{p}^{(k)}$ but vary between them, and $u_i^{(k)} \sim U(0,1)$ are independent, for each entry $\boldsymbol{p}^{(k)}(i)$. Finally, $A^{(k)}$ is the normalizing constant such that $\left\|\boldsymbol{p}^{(k)}\right\|_1 = 1$.

1. constant, $\boldsymbol{p}^{(k)}(i) \equiv A^{(k)}$
2. random, $\boldsymbol{p}^{(k)}(i) = A^{(k)} \exp(-40 u_i^{(k)})$.
3. quadratic, $\boldsymbol{p}^{(k)}(i) = A^{(k)} \exp(-30(a^{(k)} + 1){x_i}^2 + 20(2b^{(k)} - 1)x_i + 20(2c^{(k)} - 1))$ where the x_i are sequenced evenly in $[0,1]$ (that is, the first element $x_0 = 0$, and last $x_n = 1$).
4. sinusoid, $\boldsymbol{p}^{(k)}(i) = A \exp(10(3a^{(k)} + 1) \sin(x_i + b^{(k)}/10) + 10(5c^{(k)} - 4)x_i)$ where x_i are sequenced evenly from $[0, 3\pi]$.
5. "multi-scaled I", a random fifth (rounded down) of the $\boldsymbol{p}^{(k)}(i)$ are of the form $A^{(k)} \exp(-30 u_i^{(k)})$ and the remaining entries are of the form $A^{(k)} \exp(-100(u_i^{(k)} + 1))$.
6. "multi-scaled II", a random third (rounded down) of the $\boldsymbol{p}^{(k)}(i)$ are of the form $A^{(k)} \exp(-15(2a^{(k)} + 1)u_i^{(k)})$ and the remaining entries are of the form $A^{(k)} \exp[-50((2a^{(k)} + 1)u_i^{(k)} + 2b^{(k)} + 1)]$.

For each tested length n, we generated 100 random examples of each class, except for the constant class, where there is of course just one possible value. For each of these random vectors, we computed $\sum_{s \geq s_0} \boldsymbol{p}^L(s)$ to a variety of levels of accuracy γ, for

$$\begin{aligned} L &\in \left\{ 2^i \,\middle|\, i \in \{0, \ldots, 8\} \right\} \cup \left\{ \lfloor 1.5^i \rfloor \,\middle|\, i \in \{0, \ldots, 13\} \right\}, \\ s_0 &\in \left\{ \lfloor r N_L \rfloor \,\middle|\, r \in \{0.6, 0.8, 0.7, 0.85, 0.9, 0.95, 0.99\} \right\}, \end{aligned}$$

where $\lfloor x \rfloor$ is the largest integer not greater than x, and $N_L = L(n-1) + 1$ is the length of $\boldsymbol{p}^L$.

Supp. Table 2 below summarizes the results of this for the moderate accuracy levels of $\gamma = 10^{-1}$ and $\gamma = 10^{-3}$, values on the order of what users would typically require, as well as the more demanding $\gamma = 10^{-9}$, to demonstrate the flexibility of the algorithm.

	$\gamma = 10^{-9}$		$\gamma = 10^{-3}$		$\gamma = 10^{-1}$	
n	Median	Maximum	Median	Maximum	Median	Maximum
8	$7 \cdot 10^{-14}$	$9 \cdot 10^{-10}$	$2 \cdot 10^{-13}$	$9 \cdot 10^{-4}$	$2 \cdot 10^{-13}$	$9 \cdot 10^{-2}$
16	$1 \cdot 10^{-13}$	$9 \cdot 10^{-10}$	$1 \cdot 10^{-13}$	$9 \cdot 10^{-4}$	$1 \cdot 10^{-13}$	$9 \cdot 10^{-2}$
32	$1 \cdot 10^{-13}$	$9 \cdot 10^{-10}$	$1 \cdot 10^{-13}$	$9 \cdot 10^{-4}$	$1 \cdot 10^{-13}$	$9 \cdot 10^{-2}$
64	$5 \cdot 10^{-14}$	$9 \cdot 10^{-10}$	$8 \cdot 10^{-14}$	$9 \cdot 10^{-4}$	$9 \cdot 10^{-14}$	$9 \cdot 10^{-2}$
128	$5 \cdot 10^{-14}$	$9 \cdot 10^{-10}$	$5 \cdot 10^{-14}$	$9 \cdot 10^{-4}$	$5 \cdot 10^{-14}$	$9 \cdot 10^{-2}$
256	$5 \cdot 10^{-14}$	$9 \cdot 10^{-10}$	$5 \cdot 10^{-14}$	$9 \cdot 10^{-4}$	$5 \cdot 10^{-14}$	$9 \cdot 10^{-2}$
512	$5 \cdot 10^{-14}$	$9 \cdot 10^{-10}$	$4 \cdot 10^{-14}$	$9 \cdot 10^{-4}$	$4 \cdot 10^{-14}$	$8 \cdot 10^{-2}$
1024	$5 \cdot 10^{-14}$	$9 \cdot 10^{-10}$	$3 \cdot 10^{-14}$	$9 \cdot 10^{-4}$	$3 \cdot 10^{-14}$	$9 \cdot 10^{-2}$
2048	$8 \cdot 10^{-14}$	$5 \cdot 10^{-10}$	$3 \cdot 10^{-14}$	$5 \cdot 10^{-4}$	$3 \cdot 10^{-14}$	$9 \cdot 10^{-2}$
4096	$9 \cdot 10^{-14}$	$3 \cdot 10^{-11}$	$5 \cdot 10^{-14}$	$4 \cdot 10^{-4}$	$5 \cdot 10^{-14}$	$5 \cdot 10^{-2}$
8192	$1 \cdot 10^{-13}$	$2 \cdot 10^{-11}$	$5 \cdot 10^{-14}$	$8 \cdot 10^{-4}$	$5 \cdot 10^{-14}$	$8 \cdot 10^{-4}$

Table 2: The maximum and median relative errors of $\widetilde{P}^{(k,s_0,L)} = \sum_{s \geq s_0} \boldsymbol{p}^{(k)*L}(s)$ as computed by sisFFT where the $\boldsymbol{p}^{(k)}$, s_0 and L are selected as described in the text and n is the length of $\boldsymbol{p}^{(k)}$. That is, the third column is $\max_{k,L,s_0} \mathrm{rel}(\widetilde{P}^{(k,s_0,L)}_{\mathrm{sisFFT}}, \widetilde{P}^{(k,s_0,L)}_{\mathrm{NC}})$ where $\widetilde{P}^{(k,s_0,L)}_{\mathrm{x}}$ is $\widetilde{P}^{(k,s_0,L)}$ as computed by algorithm x with $x = \mathrm{NC}, \mathrm{sisFFT}$ with $\gamma = 10^{-9}$. The second column is identical, just with median in place of the max. The fourth and fifth use sisFFT with $\gamma = 10^{-3}$, and the last two columns use sisFFT with $\gamma = 10^{-1}$ The same pmfs $\boldsymbol{p}^{(k)}$ were used for every column.

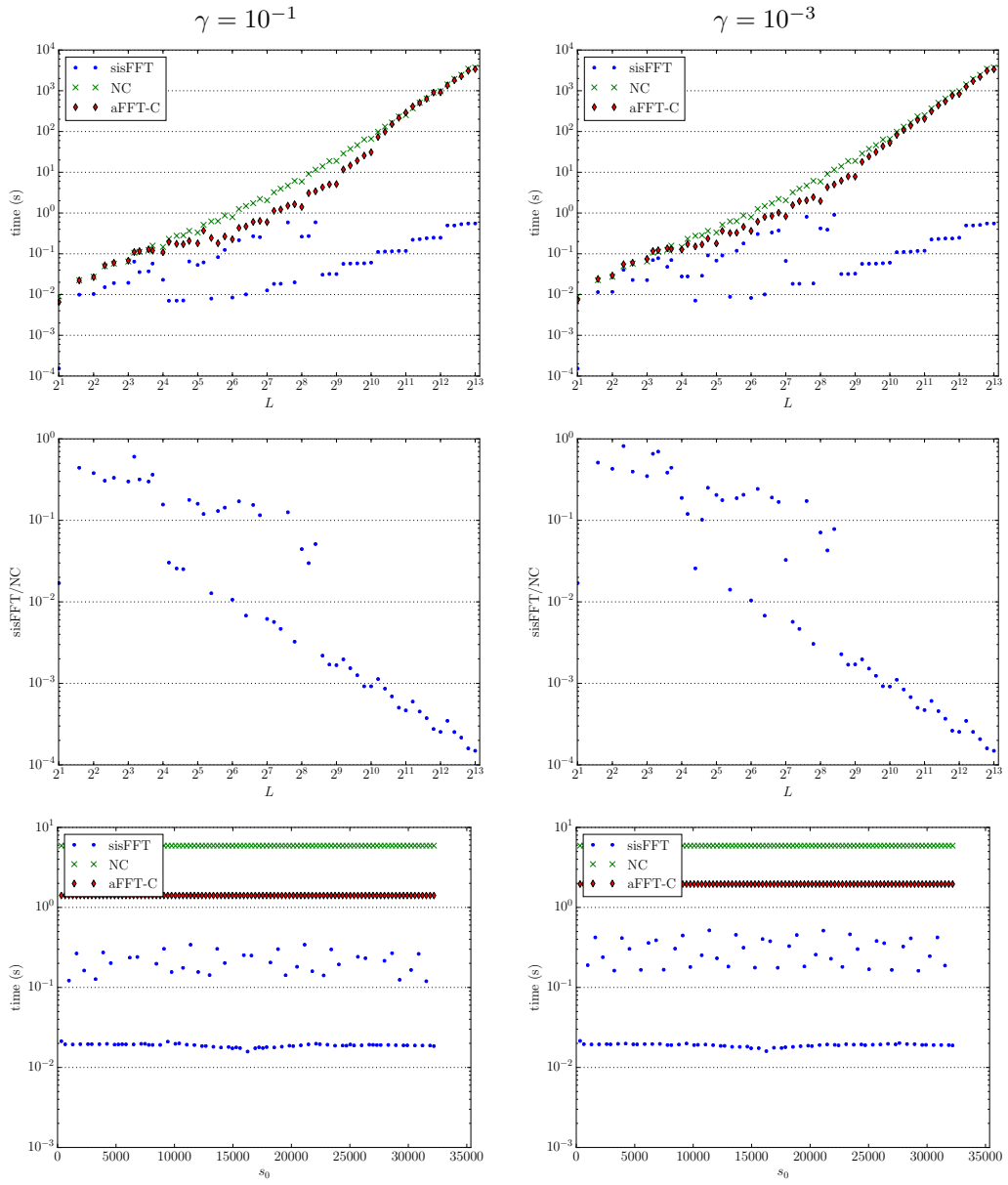

Figure 4: A comparison of timings for computing an approximation to $P = \sum_{s \geq s_0} \boldsymbol{p}^{*L}$ for various values of L using $\boldsymbol{p}$ from the second panel of Supp. Figure 2. All computations used $s_0 = \lfloor 0.95 N_L \rfloor$ where N_L is the length of $\boldsymbol{p}^{*L}$, and each convolution was performed with three different algorithms: sisFFT, NC and iterated aFFT-C (with no lower bound). The two distinct lines of sisFFT's timings: generally, comparatively slower for small L and comparatively faster for large L correspond to whether or not the sFFT shortcut is taken. The last pair of panes further explores this, showing the timing for computing P for fixed $L = 256$ with $s_0 = \lfloor 0.00 N_L \rfloor, \lfloor 0.01 N_L \rfloor, \ldots, \lfloor 0.99 N_L \rfloor$. A similar plot for smaller L have a larger proportion of the sisFFT in the higher band, while larger L, such as $L = 1024$, have all sisFFT points in the lower band. Unsurprisingly, the more relaxed accuracy bound of $\gamma = 10^{-1}$ has fewer sisFFT points in the slower band than $\gamma = 10^{-3}$.

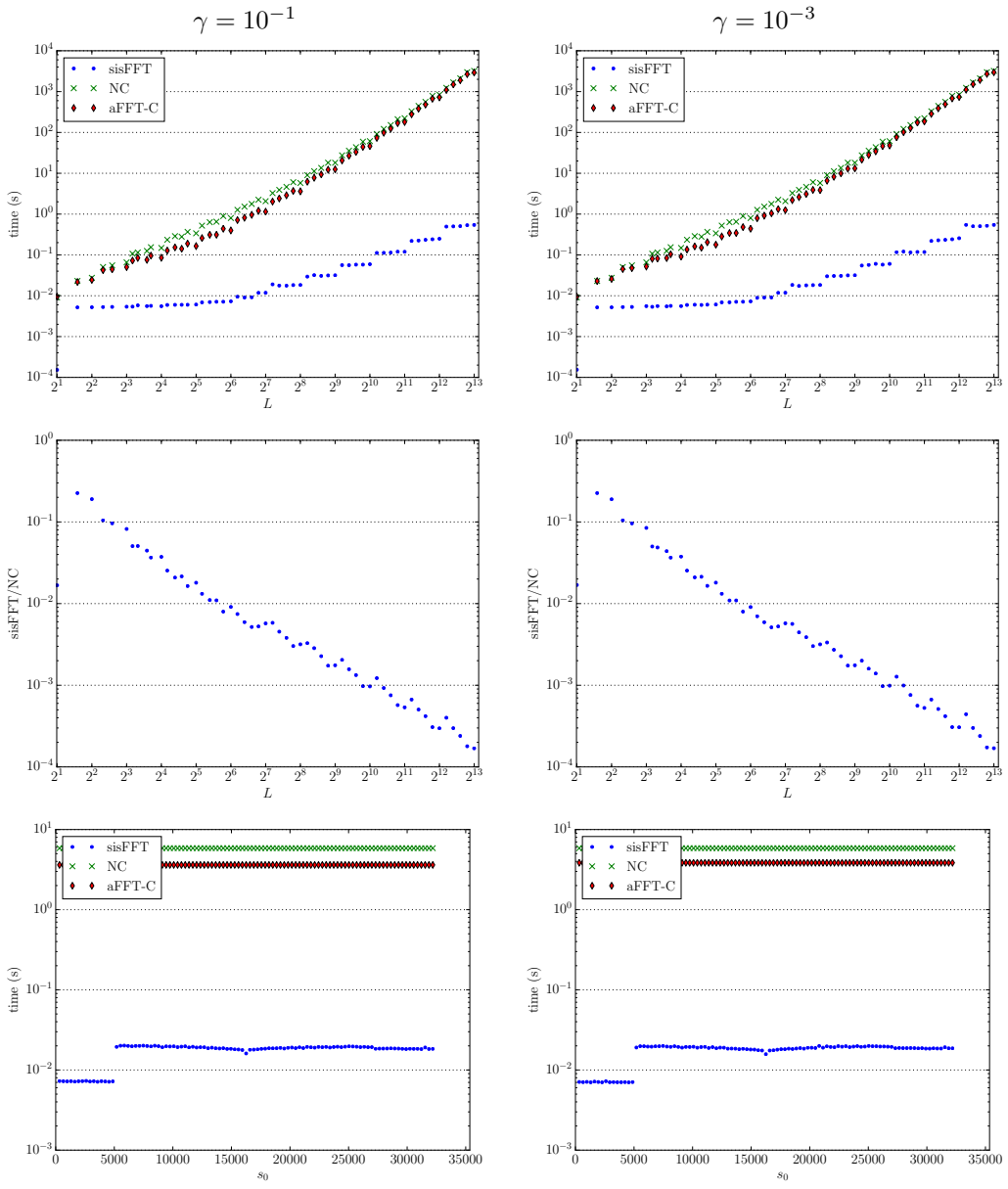

Figure 5: Same as Supp. Figure 4 but with $\boldsymbol{p}(s) = A \exp(60 \sin \frac{3\pi s}{127} - 10\frac{3\pi s}{127})$ with $s = 0, \ldots, 127$. With this pmf the sFFT shortcut always engages, allowing sisFFT to be uniformly significantly faster than both NC and aFFT-C. Similarly, the sFFT shortcut is employed for almost all the convolutions in the last pair of panes, which display the timings for fixed $L = 256$ with varying s_0 in the manner of Supp. Figure 4. For the s_0 in the initial region of approximately $s_0 < 5000$, sisFFT is able to use the FFT-C shortcut instead of requiring sFFT.

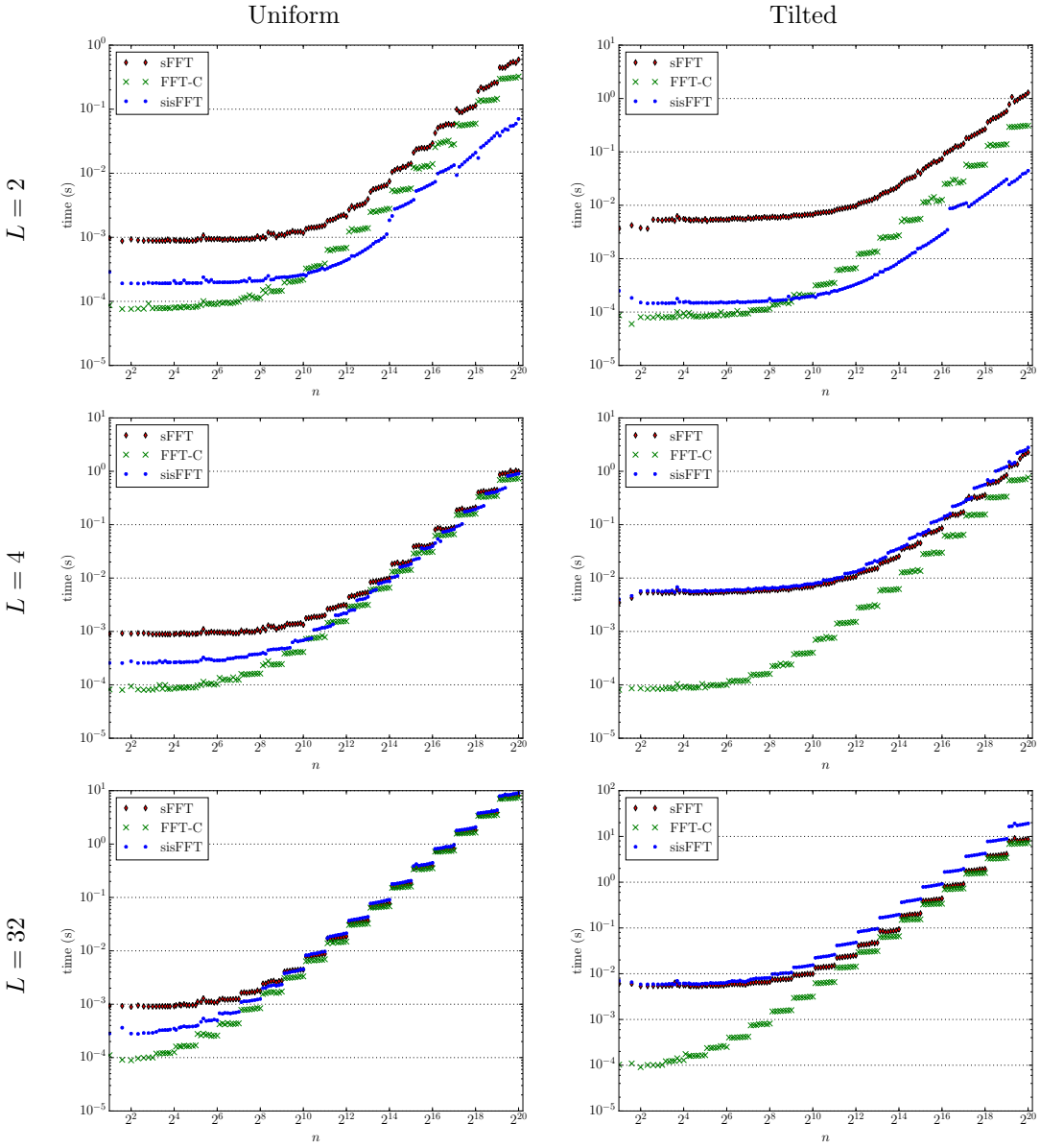

Figure 6: A comparison of sisFFT vs. sFFT vs. FFT-C for performing several computations $P = \sum_{s \geq s_0} \boldsymbol{p}^{*L}(s)$ for three values of L. The left-hand column, Uniform, uses the uniform pmf $\boldsymbol{p} = (1/n, \ldots, 1/n)$ where n is the length of that pmf, with $s_0 = \lfloor 0.5 N_L \rfloor$, which means $P \approx 0.5$ and thus FFT-C is able to calculate it accurately. The right-hand column, Tilted, is designed to require a shift to compute accurately, and thus has $\boldsymbol{p}(s) = A \exp(-30s/(n-1))$ where A is a normalising constant, and $s_0 = \lfloor 0.9 N_L \rfloor$. As can be seen, with $L = 2$, sisFFT is uniformly faster than both FFT-C and sFFT for $n > 2^{10}$, and, in the Uniform case, for $L = 4$, sisFFT is sometimes faster in the same region. Note that the FFT-C computation for Tilted is inaccurate for all displayed n and L, but both sFFT and sisFFT are accurate with only a small overhead over it.

11 Bounded aFFT-C

Algorithm 5 The BOUNDED aFFT-C algorithm, a version of aFFT-C that guarantees (18). It takes two non-negative vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ of length m and n respectively, an accuracy parameter $\alpha \geq 2$ and a lower bound $\Delta \geq 0$, and computes an approximation $\boldsymbol{c} = \widetilde{\boldsymbol{p} * \boldsymbol{q}}$, such that (18) holds with $C_{\alpha,\Delta}(\boldsymbol{p}, \boldsymbol{q}) = \boldsymbol{c}$. It is very similar to the original aFFT-C, listed in Wilson and Keich (2016), with the notable removal of the exponential shift. For more details, see Wilson (2016).

procedure Bounded aFFT-C($\boldsymbol{p}$, $\boldsymbol{q}$, α, Δ)
  Set $N = m + n - 1$ and choose $Q = 2^K \geq N$.
  Compute $\boldsymbol{c}_1, I_1 \leftarrow$ Bounded Checked FFT-C($\boldsymbol{p}, \boldsymbol{q}, \alpha, \Delta$).
  if $2CQ \log Q \geq N\,|I_1|$ **then** ▷ *Direct FFT-C is accurate for sufficiently many entries.*
    Recompute the entries $\boldsymbol{c}_1(i)$ via NC, for each $i \in I_1$.
    return $\boldsymbol{c}_1$.
  Compute the component boundaries $[b_{\boldsymbol{p},i}]_{i=1}^{n_{\boldsymbol{p}}} \leftarrow \text{PITS}(\boldsymbol{p}_{\geq\Delta}, \alpha)$ and $[b_{\boldsymbol{q},j}]_{j=1}^{n_{\boldsymbol{q}}} \leftarrow \text{PITS}(\boldsymbol{q}_{\geq\Delta}, \alpha)$. (See (Wilson and Keich, 2016, Algorithm 2).)
  if $Cn_{\boldsymbol{p}}n_{\boldsymbol{q}}Q \log Q \geq N\,|I_2|$ **then** ▷ *NC estimated to be faster.*
    Recompute the entries $\boldsymbol{c}_2(i)$ via NC, for each $i \in I_2$.
    return $\boldsymbol{c}_2$.
  else
    $\widetilde{\boldsymbol{p} * \boldsymbol{q}} = \text{PSFFT-C}(0, \boldsymbol{p}_{\geq\Delta}, [b_{\boldsymbol{p},i}], \boldsymbol{q}_{\geq\Delta}, [b_{\boldsymbol{q},j}], \alpha)$. (See (Wilson and Keich, 2016, Algorithm 3).)
    return $\widetilde{\boldsymbol{p} * \boldsymbol{q}}$.

Algorithm 6 The Bounded Checked FFT-C algorithm that gives the (18) guarantee. Given non-negative vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ and scalars $\alpha \geq 2$ and $\Delta \geq 0$, Bounded Checked FFT-C returns a vector $\boldsymbol{c} = \widetilde{\boldsymbol{p} * \boldsymbol{q}}$ as computed by FFT-C and a set I of indices such that (18) holds for $C_{\alpha,\Delta}(\boldsymbol{p}, \boldsymbol{q}) = \boldsymbol{c}$ for all $k \notin I$.

procedure Bounded Checked FFT-C($\boldsymbol{p}$, $\boldsymbol{q}$, α, Δ)
  Compute $\widetilde{\boldsymbol{p} * \boldsymbol{q}}$ via FFT-C, and note the $Q = 2^K$ of (Wilson and Keich, 2016, Lemma 2) that was used.
  Let I be the set of indices for which (Wilson and Keich, 2016, (23)) does not hold.
  if $I \neq \emptyset$ and $Q \leq Q_{\max}$ and $\boldsymbol{p}(i) < \Delta$ or $\boldsymbol{q}(i) < \Delta$ for some i **then**
    Compute $\widetilde{\boldsymbol{v}} = \widetilde{\mathbf{1}_{\boldsymbol{p}>0} * \mathbf{1}_{\boldsymbol{q}>0}}$ via FFT-C, and zero anything less than $2cK\varepsilon$ (where c is the same as defined following Lemma 1).
    Hence, by (Wilson and Keich, 2016, Corollary 2), deduce the support $I_{\text{sup}} = \text{supp}(\boldsymbol{p}_{\geq\Delta} * \boldsymbol{q}_{\geq\Delta}) = \text{supp}(\widetilde{\boldsymbol{w}})$
    Zero any entry $(\widetilde{\boldsymbol{p} * \boldsymbol{q}})(i)$ with $i \notin I_{\text{sup}}$.
    Set $I \leftarrow I \cap I_{\text{sup}}$.
  return $\widetilde{\boldsymbol{p} * \boldsymbol{q}}$, I.

References

Butler, R. (2007). *Saddlepoint Approximations with Applications*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press.

IEEE Computer Society (2008, Aug). *IEEE 754-2008, Standard for Floating-Point Arithmetic*. New York, NY: IEEE.

Wilson, H. (2016). Computing fast and accurate convolutions. Master's thesis, University of Sydney.

Wilson, H. and U. Keich (2016). Accurate pairwise convolutions of non-negative vectors via FFT. *Computational Statistics & Data Analysis 101*, 300–315.