Variance Stabilization for Noisy+Estimate Combination in Iterative Poisson Denoising

Lucio Azzari and Alessandro Foi

Abstract—We denoise Poisson images with an iterative algorithm that progressively improves the effectiveness of variance-stabilizing transformations (VST) for Gaussian denoising filters. At each iteration, a combination of the Poisson observations with the denoised estimate from the previous iteration is treated as scaled Poisson data and filtered through a VST scheme. Due to the slight mismatch between a true scaled Poisson distribution and this combination, a special exact unbiased inverse is designed. We present an implementation of this approach based on the BM3D Gaussian denoising filter. With a computational cost at worst twice that of the non-iterative scheme, the proposed algorithm provides significantly better quality, particularly at low SNR, outperforming much costlier state-of-the-art alternatives.

II. PRELIMINARIES AND MOTIVATION
Let $z$ be an observed noisy image composed of pixels $z(x)$, $x \in \Omega \subset \mathbb{Z}^2$, modeled as independent realizations of a Poisson process with parameter $y(x) \geq 0$:

$$z(x) \sim \mathcal{P}(y(x)),$$

$$\mathbb{P}(z(x) | y(x)) = \begin{cases} \frac{y(x)^{z(x)}e^{-y(x)}}{z(x)!} & \text{if } z \in \mathbb{N} \cup \{0\} \\ 0 & \text{elsewhere.} \end{cases}$$

The mean and variance of $z(x)$ coincide and are equal to $y(x)$:

$$\mathbb{E}[z(x) | y(x)] = \text{var}[z(x) | y(x)] = y(x).$$

For conciseness, henceforth we will omit $x$ from notation.

Our goal is to compute an estimate $\hat{y}$ of $y$ from $z$. To this purpose, in the archetypal VST framework, the Anscombe forward transformation $a$ [1] yields an image

$$a(z) = 2\sqrt{z + \frac{3}{8}}$$

which can be treated as corrupted by AWGN with unit variance. Thus, it can be denoised using any filter $\Phi$ designed for AWGN. If the denoising is ideal, we have

$$\Phi[a(z)] = \mathbb{E}[a(z) | y].$$

The so-called exact unbiased inverse of $a$ [2]

$$\mathcal{I}_a^\Phi : \mathbb{E}[a(z) | y] \mapsto E\{z | y\} = y,$$

is used to return the denoised image to the original range of $z$, thus yielding an estimate of $y$:

$$\hat{y} = \mathcal{I}_a^\Phi (\Phi[a(z)]).$$

However, for small $y$, when the SNR is very low, the stabilization is imprecise and the conditional distribution of $a(z)$ is far from the assumed normal, in terms of both scale and shape, leading to ineffective filtering with $\Phi$. This issue has been commonly addressed either by applying VST after binning, i.e. by stabilizing sums of adjacent pixels instead of individual pixels [3], [4], [6]–[10], or by similarly stabilizing transform coefficients [11] (essentially inserting the VST within the denoising method itself). All these stratagems aim at increasing the SNR of the data subject to the VST.

In this letter, we introduce an alternative and more direct way to improve the SNR prior to VST, by combining the noisy observation $z$ with a previously obtained estimate of the noise-free data $y$, leading to the following simple iterative algorithm.
III. PROPOSED ITERATIVE ALGORITHM

A subscript index denotes a symbol’s instance at a particular iteration, e.g., $\hat{y}_i$ is the estimate of $y$ at iteration $i$.

We initialize the algorithm by setting $\hat{y}_0 = z$. At each iteration $i = 1, \ldots, K$ we compute a convex combination of $\hat{y}_{i-1}$ and $z$

$$\tilde{z}_i = z_i (1 - \lambda_i) \hat{y}_{i-1}, \quad (1)$$

where $0 < \lambda_i \leq 1$. Provided we can treat $\hat{y}_{i-1}$ as a surrogate for $y$, we have $E[\tilde{z}_i|y] = \lambda_i^2 \text{var}(z_i|y)$; thus $\tilde{z}_i$ has higher SNR than $z$ for any $\lambda_i < 1$. We then apply a VST $f_i$ to $\tilde{z}_i$ and obtain an image $\hat{z}_i = f_i(\tilde{z}_i)$, which we denoise with a filter $F$ for AWGN to obtain a denoised image $D_i = \Phi[\hat{z}_i]$.

Assuming $D_i = E[f_i(\tilde{z}_i)|y]$, the exact unbiased inverse of $f_i$, $I_{k_i}^{h_i} : E[f_i(\hat{z}_i)|y] \rightarrow E[\tilde{z}_i|y] = y$, brings this image to the original range, yielding

$$\hat{y}_i = I_{k_i}^{h_i}(D_i),$$

which is either used for the next iteration if $i < K$, or output as final estimate $\hat{y}_K = \hat{y}$.

Let us provide further details on the above basic procedure.

A. Forward variance-stabilizing transformation

Consider the scaled variable $\lambda_i^{-2} \tilde{z}_i$ and let us model $\hat{y}_{i-1}$ as $y$. Setting $q_i(t) = \lambda_i t - \frac{1 - \lambda_i}{\lambda_i}$, the conditional probability

$$P(\lambda_i^{-2} \tilde{z}_i|y) = \begin{cases} \frac{q_i(A_{i}^{-2} \tilde{z}_i)}{q_i(A_{i}^{-2} \tilde{z}_i))} & q_i(A_{i}^{-2} \tilde{z}_i) \in \mathbb{N} \cup [0] \\ 0 & \text{elsewhere} \end{cases} \quad (2)$$

Unless $\lambda_i = 1$, this is not a Poisson distribution. However, the mean and variance of $\lambda_i^{-2} \tilde{z}_i$ do always coincide:

$$E(\lambda_i^{-2} \tilde{z}_i|y) = \text{var}(\lambda_i^{-2} \tilde{z}_i|y) = \lambda_i^{-2} y.$$

Hence, $\lambda_i^{-2} \tilde{z}_i$ resembles $P(\lambda^{-2} y)$ and indeed one can prove [12] that it is asymptotically stabilized by the Anscombe transformation $a$. Thus, we set $f_i(t) = a(\lambda_i^{-2} t)$.

B. Exact unbiased inverse transformation

The exact unbiased inverse $I_{k_i}^{h_i}$ is defined upon (2) as

$$E(f_i(\tilde{z}_i)|y) = \sum_{\tilde{z}_i = q_i(A_{i}^{-2} \tilde{z}_i))} a(\lambda_i^{-2} \tilde{z}_i) \lambda_i^{-2} \tilde{z}_i |y) = y. \quad (3)$$

We have $I_{k_i}^{h_i} \approx \lambda_i^{-2} f_i^P$, with $f_i^P = f_i^P$ [2]. The appendix describes how to accurately compute (3) in practice.

C. Binning

It is natural to combine the convex combination (1) with a linear binning; this can be especially beneficial at the first iterations, when $\hat{y}_{i-1}$ is a poor estimate of $y$. Specifically, a binning operator $B_h$ can be applied to $\hat{y}_i$, yielding a smaller image where each block (i.e. bin) of $h_i \times h_i$ pixels from $\hat{y}_i$ is replaced by a single pixel equal to their sum. $B_h$ clearly commutes with (1) and

$$B_h[\hat{y}_i] = \lambda_i B_h[\hat{y}_{i-1} + (1 - \lambda_i) \hat{y}_i].$$

Algorithm 1: Iterative Poisson Image Denoising via VST

1: $\hat{y}_0 = z$
2: for $i = 1$ to $K$ do
3: $\hat{z}_i = \lambda_i z + (1 - \lambda_i) \hat{y}_{i-1}$
4: $\hat{z}_i = f_i(B_h[\hat{z}_i])$
5: $D_i = \Phi[\hat{z}_i]$
6: $\hat{y}_i = B_{h_i}^{-1}[I_{k_i}^{h_i}(D_i)]$
7: end for
8: return $\hat{y} = \hat{y}_K$

Algorithm 2: Debinning $\hat{y}_i = B_{h_i}^{-1}[I_{k_i}^{h_i}(D_i)]$

1: $\hat{y}_{i,0} = 0$
2: for $j = 1$ to $J$ do
3: $r_j = I_{h_i}^{j}(D_i) - B_{h_i}[\hat{y}_{i,j-1}]$
4: $\hat{y}_{i,j} = \max(0, \hat{y}_{i,j-1} + \frac{\partial h_{\hat{y}_i}}{h^{-2} r_j})$
5: end for
6: return $\hat{y}_i = \hat{y}_{i,J}$

Since $B_{h_i}[z] \sim P(\hat{y}_i, z)] = P(E[\hat{y}_i, z]|y)$, and modeling again $\hat{y}_{i-1}$ as $y$, we have that $B_h(z, \hat{y}_{i-1})$ (resp. $\lambda_i^{-2} B_h(z, \hat{y}_{i-1})$) is subject to the same conditional probability of $\hat{y}_{i-1}$ (resp. $\lambda_i^{-2} \tilde{z}_i$), which means that the adoption of binning does not interfere with the subsequent VST, denoising, and inverse VST. Thus, we can define $\tilde{z}_i = f_i(B_{h_i}[\hat{y}_{i-1}])$ without modifying $f_i$.

Debinning: An inverse binning operator $B_{h_i}^{-1}$ is applied after the exact unbiased inversion,

$$\hat{y}_i = B_{h_i}^{-1}[I_{k_i}^{h_i}(D_i)],$$

returning a full-size image estimate $\hat{y}_i$ such that

$$B_{h_i}[\hat{y}_i] = I_{k_i}^{h_i}(D_i). \quad (4)$$

All the above steps are summarized in Algorithm 1 and as

$$\hat{y}_i = B_{h_i}^{-1}[I_{k_i}^{h_i}(\Phi[\hat{y}_i, B_{h_i}[\lambda_i z + (1 - \lambda_i) \hat{y}_{i-1}]])],$$

IV. IMPLEMENTATION AND RESULTS

For applications we adopt the BM3D denoising algorithm [5]; yet, other AWGN filters such as [13]–[20] may be used as well, and also lead to competitive results as shown in the supplementary material [21].

In the debinning step, to compute $B_{h_i}^{-1}[I_{k_i}^{h_i}(D_i)], I_{k_i}^{h_i}(D_i)$ is first divided by $h_i^2$, i.e. by number of pixels in the bin, and upscaled to the size of $z$ via cubic spline interpolation $U_{h_i}$. To enforce the constraint (4), the output of interpolation is recursively binned by $B_{h_i}$ and subtracted from the target $I_{k_i}^{h_i}(D_i)$, giving a residual which is upsamples and accumulated. This subroutine, summarized in Algorithm 2, is an instance of the recursive shaping regularization with nonnegativity [22], [23].

Our current implementation of Algorithm 1 is determined by four parameters: $K$ (number of iterations), $\lambda_K$, $h_1$, $h_K$ (first and last bin sizes); other values of $\lambda_i$ and $h_i$ are defined as $\lambda_i = 1 - \frac{i}{K-1}$ (1–$K$) and $h_i = \max(h_K, h_1 - 2i + 2)$. We use

\footnote{Matlab software available at http://www.cs.tut.fi/~foi/invsnce/}
decreasing $h_t$ since binning can cause loss of image details and it becomes progressively less useful when $h_t$ gets larger and the role of $\hat{y}_{i-1}$ dominates in improving the SNR of the VST input. Obviously, $B_1$ and $B_2^{-1}$ are identity operators.

The Poisson image $z$ is the only input to our algorithm; the parameters $K$, $\lambda, h_t$, $h_K$ are adaptively selected based on the quantiles of $z$, following a training [21] over 6 images not included in the experiments test dataset; we fix $J = 9$.

PSNR (dB) results of the proposed algorithm and [2]–[4], [7], as well as their versions with binning, are reported in Table I. Table II gives a separate comparison with [8], over the different dataset of 256×256 downsampled images adopted by its authors. The tables demonstrate the superior overall performance of the proposed algorithm, also confirmed by visual inspection of the examples in Figure 1.

The complexity of Algorithm 1 is dominated by the filter $\Phi$ and possibly by the debinning operators $B_2^{-1}$. The overall execution time depends especially on the number of iterations $K$ and on $h_K$, which sets the size of the largest image to be filtered by $\Phi$; all our results have $K \leq 4$. Table I and Table II report also the average execution times for 256×256 images. We ran the proposed algorithm and [2]–[4] on a single thread of a 3.4-GHz Intel i7 CPU; the runtimes for [7], [8] are taken from the respective articles, where [8] uses a 3.3-GHz Intel i7, and [7] also uses an Intel i7. The proposed algorithm and [2] are significantly less expensive than any of the other methods.

**V. DISCUSSION AND CONCLUSIONS**

We presented an iterative VST framework for Poisson denoising. The iterative combination with a previous estimate refines...
the stabilization and helps to cope with extreme low-SNR cases, in which a standard VST approach [2] underperforms even when endowed with binning.

To analyze the importance of embedding the VST framework within the iterations, in Table III we compare our results from Table I with those by a simplified version of Algorithm 1, where the VST is external to the loop: \( f_t \) and \( I^t_{\hat{a}} \) are replaced by identity operators and \( a \) and \( I^t_{a^p} \) are applied outside of the algorithm. The significant gain in the table confirms that the improvement over [2] is not a mere consequence of a better denoising due to iterative filtering at multiple scales.

Also P4IP [7] relies on iterative AWGN filtering to denoise the Poisson \( z \). In contrast to P4IP, which formulates an optimization problem to be solved upon convergence of ADMM [25] iterations, each iteration of Algorithm 1 attacks the Poisson denoising problem directly, so any \( \hat{y}_k \) can be treated as an estimate of \( y \), with \( \hat{y}_t \) already coinciding with [2]. This results in a more efficient, stable, and substantially faster procedure, where \( \Phi \) (e.g., BM3D) is used as explicit denoiser for AWGN with variance 1 set by the VST without need of empirical tuning.

### Table II

PSNR (dB) denoising results versus those reported in [8]. Averages over 5 noise realizations.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td></td>
<td>20.44</td>
<td>19.86</td>
<td>20.65</td>
<td>20.47</td>
<td>21.23</td>
<td>18.56</td>
<td>20.50</td>
<td>0.82s</td>
</tr>
<tr>
<td>MMSE est [8]</td>
<td>2</td>
<td>22.26</td>
<td>20.05</td>
<td>21.78</td>
<td>21.22</td>
<td>22.09</td>
<td>18.98</td>
<td>21.60</td>
<td>~14min</td>
</tr>
<tr>
<td>Proposed</td>
<td></td>
<td>21.93</td>
<td>20.69</td>
<td>21.46</td>
<td>21.40</td>
<td>22.32</td>
<td>19.14</td>
<td>21.62</td>
<td>0.82s</td>
</tr>
<tr>
<td>MMSE est [8]</td>
<td>4</td>
<td>23.92</td>
<td>21.60</td>
<td>22.32</td>
<td>22.26</td>
<td>21.23</td>
<td>19.56</td>
<td>22.79</td>
<td>~14min</td>
</tr>
<tr>
<td>Proposed</td>
<td></td>
<td>24.04</td>
<td>21.71</td>
<td>22.53</td>
<td>22.52</td>
<td>23.29</td>
<td>19.65</td>
<td>22.75</td>
<td>1.41s</td>
</tr>
</tbody>
</table>

### Table III

PSNR gain (average over all images in Table I) of Algorithm 1 over its simplification with external VST (see Section V).

<table>
<thead>
<tr>
<th>Peak</th>
<th>0.1</th>
<th>0.2</th>
<th>0.5</th>
<th>1</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR (dB) gain</td>
<td>0.64</td>
<td>0.66</td>
<td>0.38</td>
<td>0.22</td>
<td>0.11</td>
<td>0.13</td>
</tr>
</tbody>
</table>

The proposed algorithm achieves state-of-the-art quality in only a tiny fraction of the time required by competitive algorithms.

### Appendix: Computing the exact unbiased inverse

As in [2], we compute \( E\{a(\lambda^{-2}z_i)\} \) numerically over a finite grid of values of \( y \) and \( \lambda_t \), from which we interpolate \( I^t_{\hat{a}} \) (3) at values within the grid range. Outside of the grid range, we leverage the available implementation [2] of the exact unbiased inverse for Poisson \( I^t_{a^p} : E\{a(\lambda^{-2}z_i)\} \mapsto \lambda^{-2}y \), where \( \lambda^{-2}z_i \sim P(\lambda^{-2}y) \), through the composition

\[
E\{a(\lambda^{-2}z_i)\} \mapsto E\{a(\lambda^{-2}z_i)\} \mapsto \lambda^{-2}y \mapsto y. \tag{5}
\]

To deal with the first of the three mappings (5), we study the difference between \( E\{a(\lambda^{-2}z_i)\} \) and \( E\{a(\lambda^{-2}z_i)\} \). For \( p \sim P(\mu) \), the mean of a generic \( g(p) = 2(\sqrt{p} + d)/y \) is [1], [26]

\[
E[g(p) | \mu] = 2\sqrt{\frac{\mu + d}{y}} \left( 1 - \frac{1}{8} \frac{\mu}{(\mu + d)^2} + \frac{1}{16} \frac{\mu d}{(\mu + d)^2} - \frac{\mu}{128} \frac{3 \mu^2 + 4 \mu d}{(\mu + d)^3} + O(\mu^{-3}) \right). \tag{6}
\]

It yields \( E\{a(\lambda^{-2}z_i)\} \) when \( \mu = y, \gamma = \lambda_t, d = \frac{1}{\lambda_t} - \frac{1}{\lambda_t}y + \frac{1}{4} \lambda_t \), and \( E\{a(\lambda^{-2}z_i)\} \) when \( \mu = \lambda^{-2}y, \gamma = 1, d = \frac{3}{8} \). Then

\[
E\{a(\lambda^{-2}z_i)\} - E\{a(\lambda^{-2}z_i)\} = \frac{\lambda_t^2}{\lambda_t} \left( 1 - \frac{1}{8} \frac{\mu}{(\mu + d)^2} + \frac{1}{16} \frac{\mu d}{(\mu + d)^2} - \frac{\mu}{128} \frac{3 \mu^2 + 4 \mu d}{(\mu + d)^3} + O(\mu^{-3}) \right). \tag{7}
\]

which is however expressed as a function of \( y \), while (5) requires a function of \( E\{a(\lambda^{-2}z_i)\} \). From (6), we can approximate large \( y \) as

\[
y = \left( \frac{1}{3} E\{a(\lambda^{-2}z_i)\} \right)^2 + O(1). \tag{8}
\]

On substituting (8) into (7) we obtain

\[
E\{a(\lambda^{-2}z_i)\} - E\{a(\lambda^{-2}z_i)\} = \frac{\lambda_t^2}{\lambda_t} \left( E\{a(\lambda^{-2}z_i)\} \right)^3 + O\left( E\{a(\lambda^{-2}z_i)\} \right)^4. \tag{9}
\]

Outside of the grid range we can discard the higher-order terms from (9) and compute \( I^t_{\hat{a}}(D_t) \) using \( I^t_{\hat{a}} \) [2] as

\[
I^t_{\hat{a}}(D_t) = \lambda_t^2 I^t_{\hat{a}}(D_t + \frac{\lambda_t^2}{\lambda_t} D_t^3). \tag{10}
\]
REFERENCES


