



HoloML in Stan

Low-photon Image Reconstruction using Bayesian Tools

ISSN 2824-7795

Brian Ward  Center for Computational Mathematics, Flatiron Institute
Bob Carpenter  Center for Computational Mathematics, Flatiron Institute
David Barmherzig Center for Computational Mathematics, Flatiron Institute

Date published: 2024-02-27 Last modified: 2024-02-27

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur posuere vestibulum facilisis. Aenean pretium orci augue, quis lobortis libero accumsan eu. Nam mollis lorem sit amet pellentesque ullamcorper. Curabitur lobortis libero eget malesuada vestibulum. Nam nec nibh massa. Pellentesque porttitor cursus tellus. Mauris urna erat, rhoncus sed faucibus sit amet, venenatis eu ipsum.

Keywords: stan, coherent-diffraction-imaging, statistics

1 Contents

2	1 Introduction	2
3	1.1 Experimental setup	2
4	2 Simulating Data	2
5	2.1 Imports and helper code	2
6	2.2 Simulation parameters	3
7	3 Stan Model	6
8	3.1 Functions	6
9	3.2 Model inputs	7
10	3.3 Additional fixed information	7
11	3.4 Parameters	7
12	3.5 Model code	8
13	3.5.1 Priors	8
14	3.5.2 Likelihood	8
15	4 Optimization	8
16	4.1 Data preparation	8
17	5 Varying N_p	9
18	5.1 Prior tuning	10
19	References	10
20	Appendicies	11

¹Corresponding author: bward@flatironinstitute.org

21	6 Full Stan Code	11
22	6.1 Digression: Efficiency	13

23 1 Introduction

24 The HoloML technique is an approach to solving a specific kind of inverse problem inherent to
 25 imaging nanoscale specimens using X-ray diffraction.

26 To solve this problem in Stan, we first write down the forward scientific model given by Barmherzig
 27 and Sun, including the Poisson photon distribution and censored data inherent to the physical
 28 problem, and then find a solution via penalized maximum likelihood.

29 1.1 Experimental setup

30 In coherent diffraction imaging (CDI), a radiation source, typically an X-ray, is directed at a
 31 biomolecule or other specimen of interest, which causes diffraction. The resulting photon flux is
 32 measured by a far-field detector. The expected photon flux is approximately the squared magnitude
 33 of the Fourier transform of the electric field causing the diffraction. Inverting this to recover an
 34 image of the specimen is a problem usually known as *phase retrieval*. The phase retrieval problem is
 35 highly challenging and often lacks a unique solution (Barnett et al. 2020).

36 Holographic coherent diffraction imaging (HCDI) is a variant in which the specimen is placed some
 37 distance away from a known reference object, and the data observed is the pattern of diffraction
 38 around both the specimen and the reference. The addition of a reference object provides additional
 39 constraints on this problem, and transforms it into a linear deconvolution problem which has a
 40 unique, closed-form solution in the idealized setting (David A. Barmherzig et al. 2019).

41 The idealized version of HCDI is formulated as

- 42 Given a reference R , data $Y = |\mathcal{F}(X + R)|^2$
- 43 Recover the source image X

44 Where \mathcal{F} is an oversampled Fourier transform operator.

45 However, the real-world set up of these experiments introduces two additional difficulties. Data
 46 is measured from a limited number of photons, where the number of photons received by each
 47 detector is modeled as Poisson distributed with expectation given by Y_{ij} (referred to in the paper
 48 as *Poisson-shot noise*). The expected number of photons each detector receives is denoted N_p . We
 49 typically have $N_p < 10$ due to the damage that radiation causes the biomolecule under observation.
 50 Secondly, to prevent damage to the detectors, the lowest frequencies are removed by a *beamstop*,
 51 which censors low-frequency observations.

52 The maximum likelihood estimation of the model presented here is able to recover reasonable images
 53 even under a regime featuring low photon counts and a beamstop.

54 2 Simulating Data

55 We simulate data from the generative model directly. This corresponds to the approach taken by
 56 Barmherzig and Sun, and is based on MATLAB code provided by Barmherzig.

57 2.1 Imports and helper code

58 Generating the data requires a few standard Python numerical libraries such as `scipy` and `numpy`.
 59 `Matplotlib` is also used to simplify loading in the source image and displaying results.

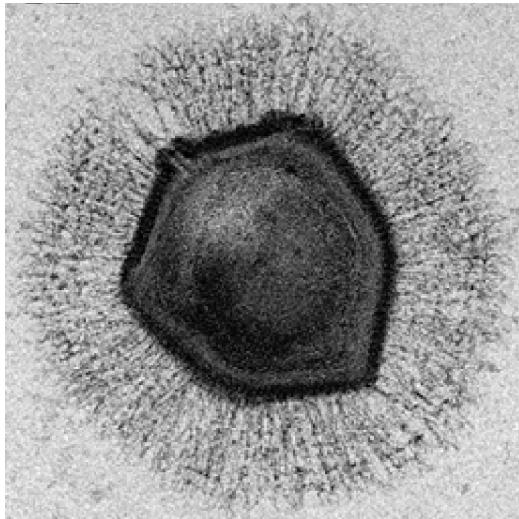
60 **2.2 Simulation parameters**

61 To match the figures in the paper (in particular, Figure 9), we use an image of size 256x256, $N_p = 1$
62 (meaning each detector is expected to receive one photon), and a beamstop of size 25x25 (correspond-
63 ing to a radius of 13), and a separation d equal to the size of the image.

64 We can then load the source image used for these simulations. In this model, the pixels of X grayscale
65 values represented on the interval $[0, 1]$. A conversion is done here from the standard RGBA encoding
66 using the above `rgb2gray` function.

67 The following is a picture of a [giant virus](#) known as a mimivirus.

68 Image credit: Ghigo E, Kartenbeck J, Lien P, Pelkmans L, Capo C, Mege JL, Raoult D., CC BY 2.5, via
69 Wikimedia Commons



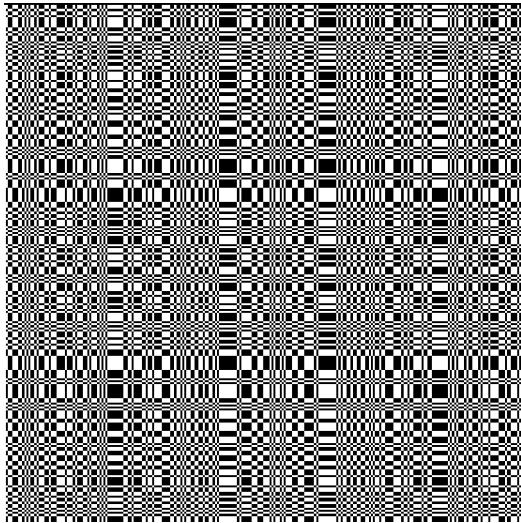
70

71 Additionally, we load in the pattern of the reference object.

72 The pattern used here is known as a *uniformly redundant array* (URA) (Fenimore and Cannon 1978).

73 It has been shown to be an optimal reference image for this kind of work, but other references
74 (including none at all) could be used with the same Stan model.

75 The code used to generate this grid is omitted from this case study. Various options such as [cappy](#)
76 exist to generate these patterns in Python.



77

78 We create the specimen-reference hybrid image by concatenating the X image, a matrix of zeros, and
79 the reference R . In the true experiment, this is done by placing the specimen some distance d away
80 from the reference, with opaque material between.

81 This distance is typically the same as the size of the specimen, N . One contribution of the HoloML
82 model is allowing recovery with the reference placed closer to the specimen, and the Stan model
83 allows for this as well.

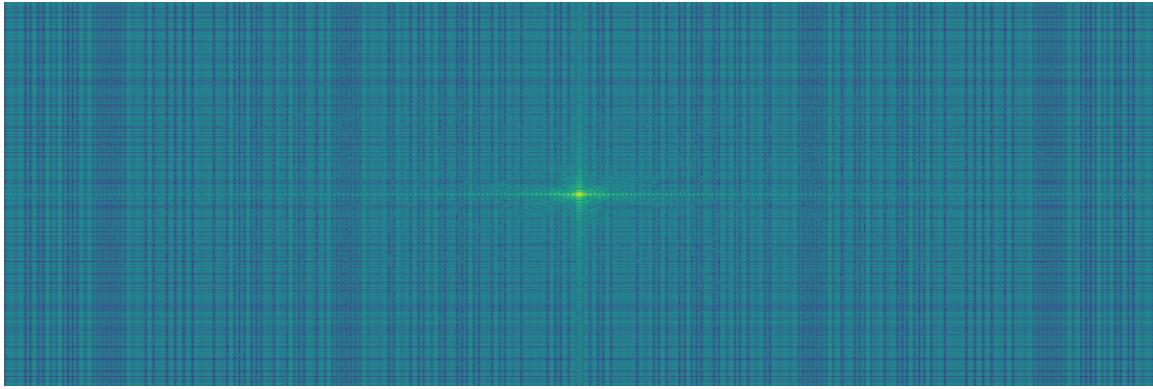
84 For this simulation we use the separation of $d = N$.



85

86 We can simulate the diffraction pattern of photons from the X-ray by taking the absolute value
87 squared of the 2-dimensional oversampled FFT of this hybrid object.

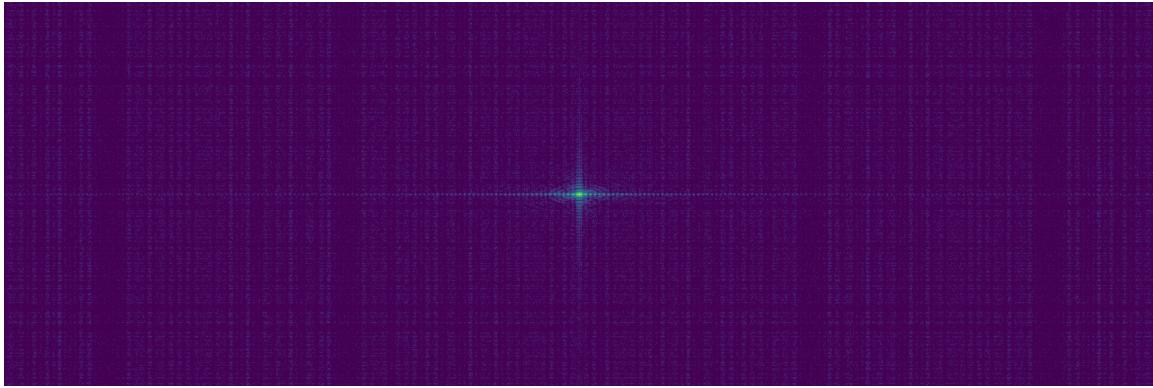
88 The oversampled FFT (denoted \mathcal{F} in the paper) corresponds to padding the image in both dimensions
89 with zeros until it is a desired size. For our case, we define the size of the padded image, M_1 by M_2 , to
90 be two times the size of our hybrid image, so the resulting FFT is twice oversampled. This is the
91 oversampling ratio traditionally used for this problem, however Barmherzig and Sun also showed
92 that this model can operate with less oversampling as well.



93

94 We simulate the photon fluxes with a Poisson pseudorandom number generator.

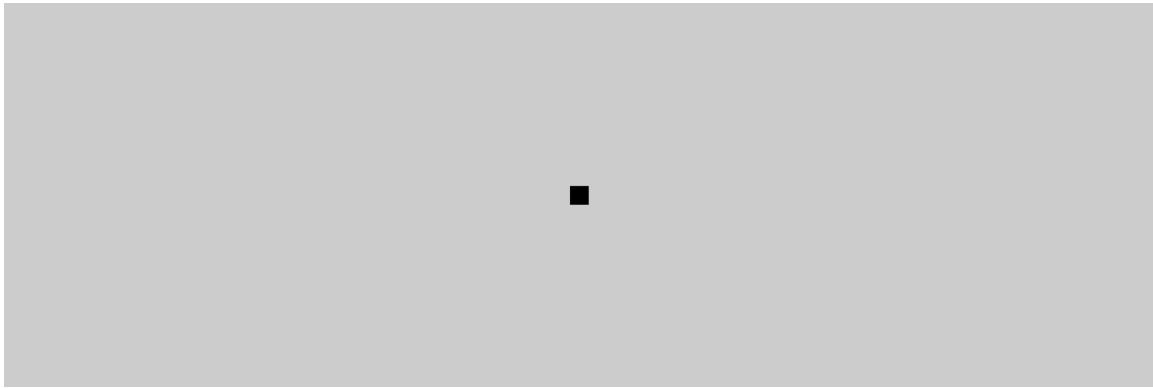
95 This code specifies a fixed seed to ensure the same fake data is generated each time.



96

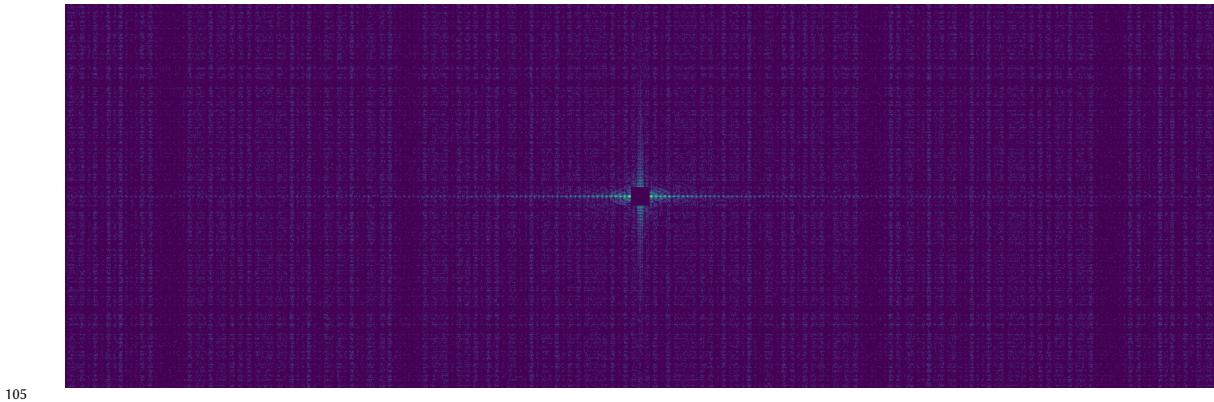
97 Finally, we need to remove the low frequency content of the data. This is caused in the physical
98 experiment by the inclusion of a beamstop, which protects the instrument used by preventing the
99 strongest parts of the beam from directly shining on the detectors.

100 The beamstop is represented by \mathcal{B} , a matrix of 0s and 1s. Zeros indicate that the data is occluded,
101 while ones represent transparent portions.



102

103 We use this matrix \mathcal{B} to mask the low frequencies of the simulated data. After removing these
104 elements from the simulated data, we have the final input which is used in our model



105

106 3 Stan Model

107 The Stan model code is a direct translation of the log density of the forward model described in the
 108 paper (David A. Barmherzig and Sun 2022) and above. The full model can be seen in the [appendix](#).

109 3.1 Functions

110 We define two helper functions to implement this model in Stan. The first is a function responsible
 111 for generating the \mathcal{B} matrix. Because Stan currently does not have FFT shifting functions, this is
 112 done by manually assigning to the corners of the matrix

```
functions {
  matrix beamstop_gen(int M1, int M2, int r) {
    matrix[M1, M2] B_cal = rep_matrix(0, M1, M2);

    // upper left
    B_cal[1 : r, 1 : r] = rep_matrix(0, r, r);
    // upper right
    B_cal[1 : r, M2 - r + 2 : M2] = rep_matrix(0, r, r - 1);
    // lower left
    B_cal[M1 - r + 2 : M1, 1 : r] = rep_matrix(0, r - 1, r);
    // lower right
    B_cal[M1 - r + 2 : M1, M2 - r + 2 : M2] = rep_matrix(0, r - 1, r - 1);
    return B_cal;
}
```

113 The FFT described in the paper is an oversampled FFT. This corresponds to embedding the image in
 114 a larger array of zeros and results in a sort of interpolation between frequencies in the result.

115 We write an overload of the `fft2` function which implements this behavior, similar to the signatures
 116 found in Matlab or Python libraries.

```
complex_matrix fft2(complex_matrix Z, int N, int M) {
  int r = rows(Z);
  int c = cols(Z);
  complex_matrix[N, M] pad = rep_matrix(0, N, M);
  pad[1 : r, 1 : c] = Z;

  return fft2(pad);
```

```

        }
} // end functions block

```

117 Note that while the first input of this function is a `complex_matrix`, it will also accept real matrices
 118 due to the built-in type promotion in Stan.

119 3.2 Model inputs

120 The Stan model needs the same information the generative model did, except it is supplied with \tilde{Y}
 121 instead of the source image X , plus a scale parameter for the prior, σ . Smaller values of σ (approaching
 122 0) lead to increasing amounts of blur in the resulting image.

```

data {
    int<lower=0> N;                      // image dimension
    matrix<lower=0, upper=1>[N, N] R;      // reference image
    int<lower=0, upper=N> d;              // separation between sample and reference image
    int<lower=N> M1;                     // rows of padded matrices
    int<lower=2 * N + d> M2;             // cols of padded matrices
    int<lower=0, upper=M1> r;            // beamstop radius. replaces omega1, omega2 in paper

    real<lower=0> N_p;                  // avg number of photons per pixel
    array[M1, M2] int<lower=0> Y_tilde; // observed number of photons

    real<lower=0> sigma;                // standard deviation of pixel prior.
}

```

123 The constraints listed above, such as `lower=0`, perform input validation. For example, the size of the
 124 padded FFT is, at a minimum, the size of the hybrid $X0R$ specimen, and we are able to encode this in
 125 the model with the lower bounds on `M1` and `M2`.

126 3.3 Additional fixed information

127 Stan provides the ability to compute transformed data, values which depend on the inputs but
 128 only need to be evaluated once per model. This allows us to construct and store \mathcal{B} once, without
 129 recomputing it each iteration or requiring it as input.

```

transformed data {
    matrix[M1, M2] B_cal = beamstop_gen(M1, M2, r);
    matrix[d, N] separation = rep_matrix(0, d, N);
}

```

130 3.4 Parameters

131 This model has only one parameter, the image X . It is constrained to grayscale values between 0 and
 132 1.

```

parameters {
    matrix<lower=0, upper=1>[N, N] X;
}

```

133 **3.5 Model code**

134 **3.5.1 Priors**

135 We add a prior on X to impose an L2 penalty on adjacent pixels. This induces a Gaussian blur on the
136 result, and it is not strictly necessary for running the model.

137 This prior is coded in our Stan program by looping over the rows and columns and using a vectorized
138 call to the `normal` distribution. This results in each pixel being adjacent to 4 others. One could also
139 formulate a prior which includes diagonally adjacent pixels

```
model {
    for (i in 1 : rows(X) - 1) {
        X[i] ~ normal(X[i + 1], sigma);
    }
    for (j in 1 : cols(X) - 1) {
        X[ :, j] ~ normal(X[ :, j + 1], sigma);
    }
}
```

140 **3.5.2 Likelihood**

141 The model likelihood encodes the forward model. We construct the hybrid specimen, compute
142 $|\mathcal{F}(X0R)|^2$, and then compute the rate λ by scaling by the average number of photons N_p .

143 We then loop over this result. If the current indices are not occluded by the beamstop \mathcal{B} , we say that
144 the data \tilde{Y} is distributed by a Poisson distribution with λ as the rate parameter.

```
// object representing specimen and reference together
matrix[N, 2 * N + d] XOR = append_col(X, append_col(separation, R));
// signal - squared magnitude of the (oversampled) FFT
matrix[M1, M2] Y = abs(fft2(XOR, M1, M2)) .^ 2;

real N_p_over_Y_bar = N_p / mean(Y);
matrix[M1, M2] lambda = N_p_over_Y_bar * Y;

for (m1 in 1 : M1) {
    for (m2 in 1 : M2) {
        if (B_cal[m1, m2] != 0) {
            Y_tilde[m1, m2] ~ poisson(lambda[m1, m2]);
        }
    }
}
} // end model block
```

145 **4 Optimization**

146 Now that we have our simulated data and our generative model, we solve the inverse problem.

147 **4.1 Data preparation**

148 We prepare a dictionary of data corresponding to the models `data` block. This is mostly reusing
149 constants defined earlier for the data simulation.

150 To run the model from Python, we instantiate it as a `CmdStanModel` object from `cmdstanpy`.

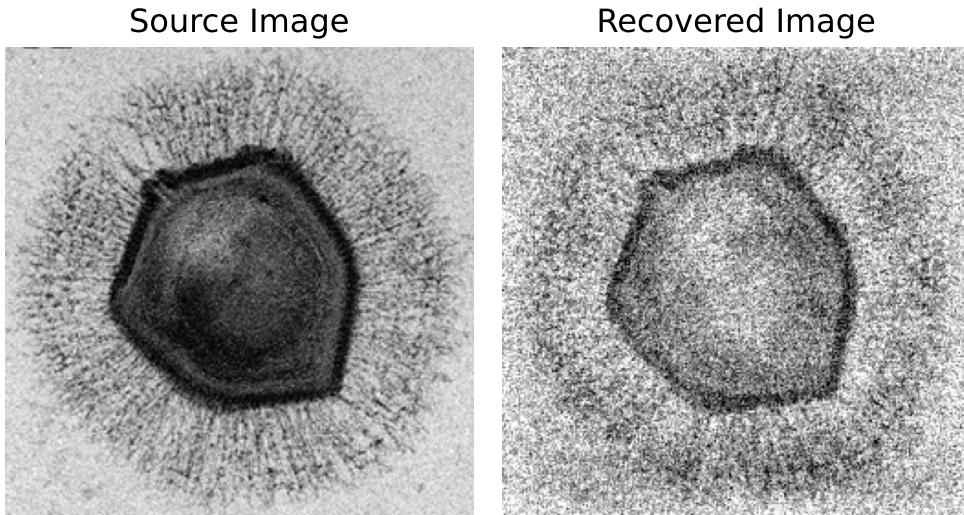
151 Here we use optimization via the limited-memory quasi-Newton L-BFGS algorithm. This method has
152 a bit more curvature information than what is available to the conjugate gradient approach, but less
153 than the second order trust-region method used in the paper. This should take a few (1-3) minutes,
154 depending on the machine you are running on.

155 It is also possible to sample the model using the No-U-Turn Sampler (NUTS), but evaluations of this
156 are out of the scope of this case study.

```
157 19:35:05 - cmdstanpy - INFO - Chain [1] start processing  
158 19:36:21 - cmdstanpy - INFO - Chain [1] done processing  
  
159 CPU times: user 634 ms, sys: 13.4 ms, total: 648 ms  
160 Wall time: 1min 16s
```

161 We use the function `stan_variable` to extract the maximum likelihood estimate (MLE) from the fit
162 object returned by optimization.

163 We can use this to plot the recovered image alongside the original.



164

165 5 Varying N_p

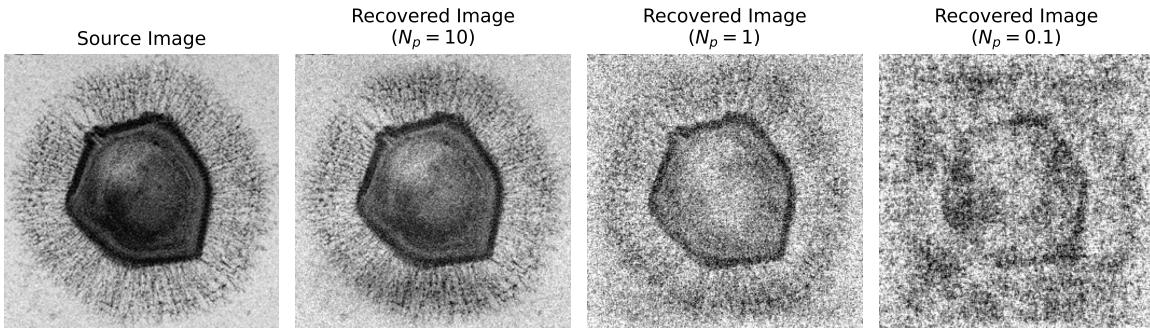
166 The above selection of $N_p = 1$ is a reasonable choice for real experiment, but both smaller and larger
167 numbers of expected photons may be used. The following are results for two other levels, $N_p = 0.1$
168 and $N_p = 10$

169 This requires repeating the final few steps of the data generation and then re-fitting the model
170 accordingly.

```
171 19:36:22 - cmdstanpy - INFO - Chain [1] start processing  
172 19:37:47 - cmdstanpy - INFO - Chain [1] done processing  
  
173 CPU times: user 634 ms, sys: 8.52 ms, total: 642 ms  
174 Wall time: 1min 24s  
  
175 19:37:48 - cmdstanpy - INFO - Chain [1] start processing  
176 19:38:27 - cmdstanpy - INFO - Chain [1] done processing  
  
177 CPU times: user 622 ms, sys: 11.7 ms, total: 634 ms  
178 Wall time: 40 s
```

179 It is worth noting that these two optimizations take very different amounts of time compared to the
180 original, as the differing amounts of data yield posteriors which are more or less normal.

181 In addition to the difference in runtime, the resulting images are very different.



182

183 5.1 Prior tuning

184 The above choice of $\sigma = 1$ has a very slight effect on the output image.

185 We also show the recovered image for $\sigma = 20$, which provides even less smoothing than the above, and
186 for $\sigma = 0.05$. This smaller value imposes a greater penalty on adjacent pixels which are significantly
187 different than each other, smoothing out the result.

188 Each of these is done with the original value of $N_p = 1$

189 19:38:29 - cmdstanpy - INFO - Chain [1] start processing

190 19:39:56 - cmdstanpy - INFO - Chain [1] done processing

191 CPU times: user 624 ms, sys: 16.6 ms, total: 641 ms

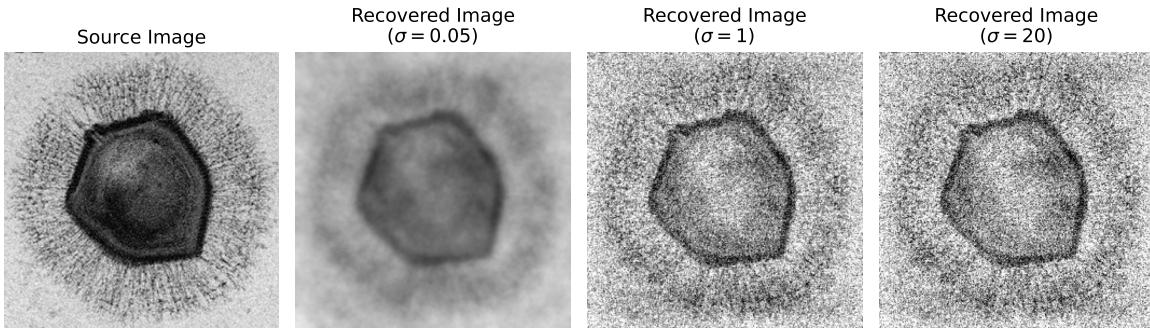
192 Wall time: 1min 28s

193 19:39:57 - cmdstanpy - INFO - Chain [1] start processing

194 19:41:23 - cmdstanpy - INFO - Chain [1] done processing

195 CPU times: user 637 ms, sys: 11.9 ms, total: 649 ms

196 Wall time: 1min 26s



197

198 References

199 Barmherzig, David A., and Ju Sun. 2022. “Towards Practical Holographic Coherent Diffraction
200 Imaging via Maximum Likelihood Estimation.” *Opt. Express* 30 (5): 6886–906. <https://doi.org/10.1364/OE.445015>.

- 202 Barmherzig, David A, Ju Sun, Po-Nan Li, T J Lane, and Emmanuel J Candès. 2019. “Holographic Phase
 203 Retrieval and Reference Design.” *Inverse Problems* 35 (9): 094001. [https://doi.org/10.1088/1361-
 204 6420/ab23d1](https://doi.org/10.1088/1361-6420/ab23d1).
- 205 Barnett, Alexander H, Charles L Epstein, Leslie F Greengard, and Jeremy F Magland. 2020. “Geometry
 206 of the Phase Retrieval Problem.” *Inverse Problems* 36 (9): 094003. [https://doi.org/10.1088/1361-
 207 6420/aba5ed](https://doi.org/10.1088/1361-6420/aba5ed).
- 208 Fenimore, E. E., and T. M. Cannon. 1978. “Coded Aperture Imaging with Uniformly Redundant
 209 Arrays.” *Appl. Opt.* 17 (3): 337–47. <https://doi.org/10.1364/AO.17.000337>.

210 **Appendices**

211 **6 Full Stan Code**

```
functions {
  /**
   * Return M1 x M2 matrix of 1 values with blocks in corners set to
   * 0, where the upper left is (r x r), the upper right is (r x r-1),
   * the lower left is (r-1 x r), and the lower right is (r-1 x r-1).
   * This corresponds to zeroing out the lowest-frequency portions of
   * an FFT.
   * @param M1 number of rows
   * @param M2 number of cols
   * @param r block dimension
   * @return matrix of 1 values with 0-padded corners
  */
  matrix beamstop_gen(int M1, int M2, int r) {
    matrix[M1, M2] B_cal = rep_matrix(1, M1, M2);
    if (r == 0) {
      return B_cal;
    }
    // upper left
    B_cal[1 : r, 1 : r] = rep_matrix(0, r, r);
    // upper right
    B_cal[1 : r, M2 - r + 2 : M2] = rep_matrix(0, r, r - 1);
    // lower left
    B_cal[M1 - r + 2 : M1, 1 : r] = rep_matrix(0, r - 1, r);
    // lower right
    B_cal[M1 - r + 2 : M1, M2 - r + 2 : M2] = rep_matrix(0, r - 1, r - 1);
    return B_cal;
  }

  /**
   * Return the matrix corresponding to the fast Fourier
   * transform of Z after it is padded with zeros to size
   * N by M
   * When N by M is larger than the dimensions of Z,
   * this computes an oversampled FFT.
   *
   * @param Z matrix of values
   * @param N number of rows desired (must be >= rows(Z))
  */
}
```

```

* @param M number of columns desired (must be >= cols(Z))
* @return the FFT of Z padded with zeros
*/
complex_matrix fft2(complex_matrix Z, int N, int M) {
    int r = rows(Z);
    int c = cols(Z);
    if (r > N) {
        reject("N must be at least rows(Z)");
    }
    if (c > M) {
        reject("M must be at least cols(Z)");
    }

    complex_matrix[N, M] pad = rep_matrix(0, N, M);
    pad[1 : r, 1 : c] = Z;

    return fft2(pad);
}
}

data {
    int<lower=0> N; // image dimension
    matrix<lower=0, upper=1>[N, N] R; // registration image
    int<lower=0, upper=N> d; // separation between sample and registration image
    int<lower=N> M1; // rows of padded matrices
    int<lower=2 * N + d> M2; // cols of padded matrices
    int<lower=0, upper=M1> r; // beamstop radius. replaces omega1, omega2 in paper

    real<lower=0> N_p; // avg number of photons per pixel
    array[M1, M2] int<lower=0> Y_tilde; // observed number of photons

    real<lower=0> sigma; // standard deviation of pixel prior.
}
transformed data {
    matrix[M1, M2] B_cal = beamstop_gen(M1, M2, r);
    matrix[d, N] separation = rep_matrix(0, d, N);
}
parameters {
    matrix<lower=0, upper=1>[N, N] X;
}
model {
    // prior - penalizing L2 on adjacent pixels
    for (i in 1 : rows(X) - 1) {
        X[i] ~ normal(X[i + 1], sigma);
    }
    for (j in 1 : cols(X) - 1) {
        X[ :, j] ~ normal(X[ :, j + 1], sigma);
    }

    // likelihood
    // object representing specimen and reference together
}

```

```

matrix[N, 2 * N + d] XOR = append_col(X, append_col(separation, R));
// signal - squared magnitude of the (oversampled) FFT
matrix[M1, M2] Y = abs(fft2(XOR, M1, M2)) .^ 2;

real N_p_over_Y_bar = N_p / mean(Y);
matrix[M1, M2] lambda = N_p_over_Y_bar * Y;

for (m1 in 1 : M1) {
    for (m2 in 1 : M2) {
        if (B_cal[m1, m2] != 0) {
            Y_tilde[m1, m2] ~ poisson(lambda[m1, m2]);
        }
    }
}
}

```

212 **6.1 Digression: Efficiency**

213 The model above is coded for readability and sticks closely to the mathematical formulation of the
 214 process. However, this does lead to an inefficient condition inside the tightest loop of the model to
 215 handle the beamstop occlusion.

216 In practice, it is possible to avoid this conditional by changing how the data is stored. Instead of
 217 storing the beamstop occlusion as a parallel matrix, we can pre-compute the list of indices which are
 218 included once and store it. Then, we can create flat representations of both the data \tilde{Y} and the rate λ ,
 219 allowing us to use a vectorized version of the Poisson distribution.

```

transformed data {
    array[M1, M2] int B_cal = beamstop_gen(M1, M2, r);
    int total = sum(to_array_1d(B_cal));
    array[total, 2] idxs;
    // pre-compute indices
    int current = 1;
    for (n in 1:M1){
        for (m in 1:M2){
            if (B_cal[n, m]){
                idxs[current, :] = {n,m};
                current += 1;
            }
        }
    }
    // flatten data accordingly
    array[total] int<lower=0> Ys;
    for (n in 1:total) {
        Ys[n] = Y_tilde[idxs[n, 1], idxs[n, 2]];
    }
}
model {
    // ... same code for computing matrix[M1, M2] lambda here
    array[total] real lambdas;
    for (n in 1:total) {

```

```
    lambdas[n] = lambda[idxs[n, i], idxs[n, j]]; // much cheaper than branching
}

Ys ~ poisson(lambdas); // fully vectorized
}
```

- 220 This formulation of the model reduces the amount of time per gradient evaluation by 15-20%. A brief
221 evaluation suggests however that the impact on optimization runtime is minimal.