Polynomial Accelerated Gibbs Sampling

Dongwuk Kim, Minji Kim and Minwoo Kim

Seoul National University

December 11, 2019

Table of Contents

Introduction

- *Our problem* is *sampling from normal distirubions.*
- Gibbs sampler is commonly used because of simple implementaion
- However, the Gibbs sampler is not efficient for a massive models.
- C. Fox and A.Parker(2017) proposed a generalized and accelerated Gibbs samplers which is fast for high-dimensional normal distributions.
- In this presentation, we review the paper of C. Fox and A. Parker(2017) - *"Accelerated Gibbs sampling of normal distributions using matrix splittings and polynomials"* - Bernoulli 23.4B and show the reseult of the proposed algorithm re-written by *Julia*.

Table of Contents

Problem

1. We want to compute a sample

$$\mathbf{y} \sim \mathbf{N}(0, A^{-1})$$

, where A is a given precision matrix.

2. An iterative samplers, such as *Gibbs*, are good option when the dimension is high becuase of it's inexpensive cost per itertion and small computer memroy requirements.
3. If the precision matrix A is sparse with $\mathcal{O}(n)$ non-zero elements, iterative methods cost only about $2n$ flops per iteration.
4. Comparable methods for non-iterative methods is sparse Cholesky factorizations($A = BB^T$ and compute $B^{-T}z$, where $z \sim N(0, I)$). However, when the bandwidth is $\mathcal{O}(n)$, then the cost is about $\mathcal{O}(n^3)$.
5. So, our goal is find an Iterative method which converges in small iterations, for example, significantly less than $\mathcal{O}(n^2)$ iterations.

Gibbs Sampling from a normal distribution

Algorithm 1: Component-sweep Gibbs sampling using a precision matrix $\mathbf{A}$

Input: Precision matrix $\mathbf{A}$, initial state $\mathbf{y}^{(0)} = (y_1^{(0)}, y_2^{(0)}, \ldots, y_n^{(0)})^T$

, and maximum iteration $k_{\max}$

Output: $\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)} \ldots, \mathbf{y}^{(k_{\max})}$, where $\mathbf{y}^{(k)} \xrightarrow{\mathcal{D}} \mathbf{N}(0, \mathbf{A}^{-1})$

for $k = 1, 2, \ldots, k_{\max}$ **do**

  for $i = 1, 2, \ldots, n$ **do**

    sample $z \sim \mathbf{N}(0, 1)$;

    $y_i^{(k)} = \frac{z}{\sqrt{a_{ii}}} - \frac{1}{a_{ii}}(\sum_{j>i} a_{ij} y_j^{(k-1)} - \sum_{j<i} a_{ij} y_j^{(k)})$

  end

end

The iteration can be written in the matrix form,

$$\mathbf{y}^{(k+1)} = -\mathbf{D}^{-1}\mathbf{L}\mathbf{y}^{(k+1)} - \mathbf{D}^{-1}\mathbf{L}^T\mathbf{y}^{(k)} + \mathbf{D}^{-1/2}\mathbf{z}^{(k)} \tag{1}$$

, where $\mathbf{z}^{(k)} \sim N(0, \mathbf{I})$, $\mathbf{D} = \text{diag}(\mathbf{A})$, and $\mathbf{L}$ is the strictly lower triangluar part.

Splitting iterative linear solvers

The iterative method for solving $\mathbf{Ax} = \mathbf{b}$ split the matrix $\mathbf{A} = \mathbf{M} - \mathbf{N}$, where $\mathbf{M}$ is invertible and easy to invert. Then $\mathbf{Ax} = \mathbf{Mx} - \mathbf{Nx} = \mathbf{b}$ or

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x} - \mathbf{M}^{-1}\mathbf{b}$$

Thus a solution to $\mathbf{Ax} = \mathbf{b}$ is a fixed point of iteration

$$\mathbf{x}^{(k+1)} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}^{(k)} - \mathbf{M}^{-1}\mathbf{b}. \tag{2}$$

The iterative solution is convergent(i.e., $\mathbf{x}^{(k)} \to \mathbf{A}^{-1}\mathbf{b}$) if and only if

$$\rho(\mathbf{M}^{-1}\mathbf{N}) < 1$$

,where $\rho(\cdot)$ is the spectral radius of a matrix. Let the error at step k be $\mathbf{e}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{A}^{-1}\mathbf{b}$. Then it's well known(e.g. *Axelsson, Iterative Solution Methods*) that

$$\lim_{k\to\infty} \left(\frac{\|\mathbf{e}^{(k+1)}\|_2}{\|\mathbf{e}^{(0)}\|_2} \right)^{1/k} = \rho(\mathbf{M}^{-1}\mathbf{N}) \tag{3}$$

Splitting iterative linear solvers

For a symmetric matrix $\mathbf{A}$, let $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{L}^T$, where $\mathbf{L}$ is a strictly lower triangular part and $\mathbf{D}$ is a diagonal part.

Splitting	$\mathbf{M}$	Convergence
Richardson	$\frac{1}{\omega}\mathbf{I}$	$0 < \omega < \frac{2}{\rho(\mathbf{A})}$
Jacobi	$\mathbf{D}$	$\mathbf{A}$ strictly diagonally dominant
Gauss-Seidel(GS)	$\mathbf{D} + \mathbf{L}$	always
SOR	$\frac{1}{\omega}\mathbf{D} + \mathbf{L}$	$0 < \omega < 2$
SSOR	$\frac{\omega}{2-\omega}\mathbf{M}_{\text{SOR}}\mathbf{D}^{-1}\mathbf{M}_{\text{SOR}}^T$	$0 < \omega < 2$

For the *Gauss-Seidel* method, (2) becomes

$$\mathbf{x}^{(k+1)} = -\mathbf{D}^{-1}\mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{D}^{-1}\mathbf{L}^T\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}. \tag{4}$$

Note that the gibbs sampler iteration (1) and linear solver's (4) are almost same.

Equivalence of linear solvers and Gibbs samplers

Theorem (Equivalence of iterative linear solvers and Gibbs samplers)

Let $\mathbf{A} = \mathbf{M} - \mathbf{N}$ *be a splitting with* $\mathbf{M}$ *invertible, and let* $\pi(\cdot)$ *be some fixed probability distribution with zero mean and fixed non-zero covariance. For any fixed vector* $\mathbf{b}$*, and random vectors* $\mathbf{c}^{(k)} \overset{i.i.d.}{\sim} \pi$*,* $k = 0, 1, 2, \ldots$*, the stationary linear iteration*

$$\mathbf{x}^{(k+1)} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b} \tag{5}$$

converges, with $\mathbf{x}^{(k)} \to \mathbf{A}^{-1}\mathbf{b}$ *as* $k \to \infty$ *whatever the initial vector* $\mathbf{x}^{(0)}$*, if and only if there exists a distribution* Π *such that the stochastic iteration*

$$\mathbf{y}^{(k+1)} = \mathbf{M}^{-1}\mathbf{N}\mathbf{y}^{(k)} + \mathbf{M}^{-1}\mathbf{c}^{(k)} \tag{6}$$

converges in distribution on Π*, with* $\mathbf{y}^{(k)} \xrightarrow{\mathcal{D}} \Pi$ *as* $k \to \infty$ *whatever the initial state* $\mathbf{y}^{(0)}$*.*

Sampling from normal distributions using matrix splittings

Theorem (Convergence of first and second moments)

Let $\mathbf{A}$ *be SPD,* $\mathbf{A} = \mathbf{M} - \mathbf{N}$ *be a convergent splitting,* μ *a fixed vector, and* $\pi(\cdot)$ *a fixed probability distribution with finite mean* ν *and non-zero covariance* $\mathbf{V}$*.*

Consider the stochastic iteration (6) where $\mathbf{c}^{(k)} \overset{i.i.d.}{\sim} \pi$*,* $k = 0, 1, 2, \ldots$*. Then, whatever the starting state* $\mathbf{y}^{(0)}$*, the following are equivalent:*

1. $\mathbf{E}[\mathbf{c}^{(k)}] = \nu$ *and* $\mathbf{Var}(\mathbf{c}^{(k)}) = \mathbf{V} = \mathbf{M}^T + \mathbf{N}$*;*
2. *the iterates* $\mathbf{y}^{(k)}$ *converges in distribution to some distribution* Π *that has mean* $\mu = \mathbf{A}^{-1}\nu$ *and covariance matrix* $\mathbf{A}^{-1}$*; in particular* $\mathbf{E}[\mathbf{y}^{(k)}] \to \mu$ *and* $\mathbf{Var}(\mathbf{y}^{(k)}) \to \mathbf{A}^{-1}$ *as* $k \to \infty$*.*

- This theorem shows *how to design* the noise distribution π so that the limit distribution Π has a desired mean μ and covariance $\Sigma = \mathbf{A}^{-1}$.
- If we set $\pi = \mathrm{N}(\nu, \mathbf{V})$ then the following are equivalent: (i) $\mathbf{V} = \mathbf{M}^T + \mathbf{N}$; (ii) $\mathbf{y}^{(k)} \xrightarrow{\mathcal{D}} \mathbf{N}(\mu, \mathbf{A}^{-1})$ where $\mu = \mathbf{A}^{-1}\nu$.

Sampling from normal distributions using matrix splittings

Algorithm 2: Stationary sampler of $\mathbf{N}(0, \mathbf{A}^{-1})$

Input: SPD precision matrix $\mathbf{A} = \mathbf{M} - \mathbf{N}$ to be convergent splitting , initial state $\mathbf{y}^{(0)}$ and maximum iteration $k_{\max}$

Output: $\mathbf{y}^{(k)}$ approximately distributed as $\mathbf{N}(0, \mathbf{A}^{-1})$

for $k = 0, 1, \ldots, k_{\max}$ **do**

sample $z \sim \mathbf{N}(0, \mathbf{M}^T + \mathbf{N})$;

$\mathbf{y}^{(k+1)} = \mathbf{M}^{-1}(\mathbf{N}\mathbf{y}^{(k)} + \mathbf{c}^{(k)})$

end

Sampling from normal distributions using matrix splittings

Algorithm 3: SSOR sampling from $\mathbf{N}(0, \mathbf{A}^{-1})$

Input: The SOR splitting $\mathbf{M}, \mathbf{N}$ of $\mathbf{A}$, relaxation parameter ω , initial state $\mathbf{y}$ and maximum iteration $k_{\max}$

Output: $\mathbf{y}$ approximately distributed as $\mathbf{N}(0, \mathbf{A}^{-1})$

set $\gamma = (\frac{2}{\omega} - 1)^{1/2}$;

for $k = 1, \ldots, k_{\max}$ **do**
  sample $\mathbf{z} \sim \mathbf{N}(0, \mathbf{I})$;
  $\mathbf{x} \leftarrow \mathbf{M}^{-1}(\mathbf{N}\mathbf{y} + \gamma \mathbf{D}^{1/2}\mathbf{Z})$;
  sample $\mathbf{z} \sim \mathbf{N}(0, \mathbf{I})$;
  $\mathbf{y} \leftarrow \mathbf{M}^{-T}(\mathbf{N}^T\mathbf{x} + \gamma \mathbf{D}^{1/2}\mathbf{Z})$

end

Table of Contents

Accleration of linear solvers by polynomials

For a set of *acceleration parameters* $\{\{\alpha_k\}, \{\tau_k\}\}$, let's introduce the second order iteration

$$\mathbf{x}^{(k+1)} = (1-\alpha_k)\mathbf{x}^{(k-1)} + \alpha_k \mathbf{x}^{(k)} + \alpha_k \tau_k \mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}). \tag{7}$$

At the first step, $\alpha_0 = 1$ and $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \tau_0 \mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)})$. Next, generate a (k+1)st order polynomial P_{k+1} recursively as

$$P_{k+1}(\lambda) = (\alpha_k - \alpha_k \tau_k \lambda) P_k(\lambda) + (1-\alpha_k) P_{k-1}(\lambda). \tag{8}$$

Then the k-step error $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{A}^{-1}\mathbf{b}$ may be written as

$$\mathbf{e}^{(k+1)} = P_k(\mathbf{M}^{-1}\mathbf{A})\mathbf{e}^{(0)} \tag{9}$$

, which can be compared directly to (5).

Accleration of linear solvers by polynomials

When estimates of the extreme eigenvalues $\lambda_{\min}$, $\lambda_{\max}$ of $\mathbf{M}^{-1}\mathbf{A}$ are available($\lambda_{\min}$, $\lambda_{\max}$ are real when $\mathbf{M}, \mathbf{N}$ are symmetric), then following coefficients $\{\{\alpha_k\}, \{\tau_k\}\}$ generate the *scaled Chebyshev* polynomials

$$\tau_k = \frac{2}{\lambda_{\max} + \lambda_{\min}}, \quad \beta_k = \left(\frac{1}{\tau_k} - \beta_{k-1}\left(\frac{\lambda_{\max} - \lambda_{\min}}{4}\right)^2\right)^{-1}, \quad \alpha_k = \frac{\beta_k}{\tau_k} \tag{10}$$

, where $\alpha_0 = 1$ and $\beta_0 = \tau_0$.

- We denote the k-th oreder Chebyshev polynomial as $\mathcal{Q}_k$.
- Note that these parameters are independent of the iterates $\{\mathbf{x}^{(k)}\}$.
- $\mathbf{M}$ is required to be symmetric, applying Chebyshev acceleration to SSOR is a common paring.

Acceleration of linear solvers by polynomials

From *Axelsson*,

$$\mathcal{Q}_k = \underset{P_k \in \mathbb{P}_k}{\operatorname{argmin}} \left(\max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |P_k(\lambda)| \right), \tag{11}$$

where $\mathbb{P}_k$ is a space of kth order polynomials. The optimal value is

$$\max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |\mathcal{Q}_k(\lambda)| = \frac{2\sigma^k}{1+\sigma^{2k}}, \quad \text{where } \sigma = \frac{1-\sqrt{\lambda_{\min}/\lambda_{\max}}}{1+\sqrt{\lambda_{\min}/\lambda_{\max}}} \in [0,1). \tag{12}$$

From (9), $\mathbf{e}^{(k+1)} = \mathcal{Q}_k(\mathbf{M}^{-1}\mathbf{A})\mathbf{e}^{(0)}$ and then the asymptotic convergence factor is bounded above by

$$\lim_{k \to \infty} \{\max |\mathcal{Q}_k(\lambda)|\}^{1/k} = \sigma \tag{13}$$

Axelsson also shows the convergence factor of non-accelerated iterative solver is bounded below by $\rho = \frac{1-\lambda_{\min}/\lambda_{\max}}{1+\lambda_{\min}/\lambda_{\max}}$. Then $\sigma < \rho$ always holds, so the Chebyshev acceleration *really accelerate* a calculation procedures.

Acceleration of Gibbs sampling by polynomials

- By slightly chainging (7), we can consider the second order stochastic iteration

$$\mathbf{y}^{(k+1)} = (1-\alpha_k)\mathbf{y}^{(k-1)} + \alpha_k \mathbf{y}^{(k)} + \alpha_k \tau_k \mathbf{M}^{-1}(\mathbf{c}^{(k)} - \mathbf{A}\mathbf{y}^{(k)}). \qquad (14)$$

  Only diffrenet thing is the vector $\mathbf{b}$ has been replaced by a random vector $\mathbf{c}^{(k)}$.
- Next, introduce some coefficients a_k, b_k, κ_k defined by $a_k = (2-\tau_k)/\tau_k + (b_k - 1)(1/\tau_k + 1/\kappa_k - 1)$, $b_k = 2\kappa_k(1-\alpha_k)/(\alpha_k\tau_k) + 1$, $\kappa_{k+1} = \alpha_k\tau_k + (1-\alpha_k)\kappa_k$, and $\kappa_1 = \tau_0$.
- Also, suppose that $\{\{\alpha_k\}, \{\tau_k\}\}$ that are independent of $\{\mathbf{x}^{(k)}\}$.
- Then (14) *converges in distribution to our targe distribution* if the polynomial accerated linear solver converges.

Main Results

Theorem (Accelerated Gibbs sampler)

Let $\mathbf{A}$ *be SPD and* $\mathbf{A} = \mathbf{M} - \mathbf{N}$ *be a symmetric splitting. Define an noise vectors* $\mathbf{c}^{(k)} \overset{ind.}{\sim} (\nu, a_k\mathbf{M} + b_k\mathbf{N})$.

1. *If the accelerated linear solver (7) converges to* $\mathbf{A}^{-1}\mathbf{b}$ *then the accelerated stochastic iteration (14) converges to a distribution with moments* $(\mathbf{A}^{-1}\nu, \mathbf{A}^{-1})$. *Furthermore, if the* $\{\mathbf{c}^{(k)}\}$ *are normal, then*

$$\mathbf{y}^{(k)} \xrightarrow{\mathcal{D}} \mathbf{N}(\mu = \mathbf{A}^{-1}\nu, \mathbf{A}^{-1}). \tag{15}$$

2. $\mathbf{E}(\mathbf{y}^{(k)}) - \mathbf{A}^{-1}\nu = P_k(\mathbf{M}^{-1}\mathbf{A})(\mathbf{E}(\mathbf{y}^{(0)}) - \mathbf{A}^{-1}\nu) \to 0$
3. $\mathbf{Var}(\mathbf{y}^{(k)}) - \mathbf{A}^{-1} = P_k(\mathbf{M}^{-1}\mathbf{A})(\mathbf{Var}(\mathbf{y}^{(0)}) - \mathbf{A}^{-1})P_k(\mathbf{M}^{-1}\mathbf{A})^T \to 0.$

- Chebyshev polynomial accelerated normal sampler is guaranteed to converge faster than any other acceleration scheme that has the parameters $\{\alpha_k, \tau_k\}$ independent of the iterates $\{\mathbf{y}^{(k)}\}$.

Algorithm: Chebyshev accelerated Gibbs sampler

Algorithm 1: Chebyshev accelerated SSOR sampling from $\mathrm{N}(\mathbf{0}, \mathbf{A}^{-1})$

input : The SSOR splitting $\mathbf{M}$, $\mathbf{N}$ of $\mathbf{A}$; smallest eigenvalue $\lambda_{\min}$ of $\mathbf{M}^{-1}\mathbf{A}$; largest eigenvalue $\lambda_{\max}$ of $\mathbf{M}^{-1}\mathbf{A}$; relaxation parameter ω; initial state $\mathbf{y}^{(0)}$; $k_{\max}$

output: $\mathbf{y}^{(k)}$ approximately distributed as $N(\mathbf{0}, \mathbf{A}^{-1})$

set $\gamma = \left(\frac{2}{\omega} - 1\right)^{1/2}$, $\delta = \left(\frac{\lambda_{\max} - \lambda_{\min}}{4}\right)^2$, $\theta = \frac{\lambda_{\max} + \lambda_{\min}}{2}$;

set $\alpha = 1$, $\beta = 2/\theta$, $\tau = 1/\theta$, $\tau_c = \frac{2}{\tau} - 1$;

for $k = 1, \ldots, k_{\max}$ **do**

  if $k = 0$ **then**

    $b = \frac{2}{\alpha} - 1, \quad a = \tau_c b, \quad \kappa = \tau;$

  else

    $b = 2(1-\alpha)/\beta + 1, \quad a = \tau_c + (b-1)\left(1/\tau + 1/\kappa - 1\right), \quad \kappa = \beta + (1-\alpha)\kappa;$

  end

  sample $\mathbf{z} \sim \mathrm{N}(\mathbf{0}, \mathbf{I})$;

  $\mathbf{c} = \gamma b^{1/2} \mathbf{D}^{1/2} \mathbf{z}$;

  $\mathbf{x} = \mathbf{y}^{(k)} + \mathbf{M}^{-1}(\mathbf{c} - \mathbf{A}\mathbf{y}^{(k)})$;

  sample $\mathbf{z} \sim \mathrm{N}(\mathbf{0}, \mathbf{I})$;

  $\mathbf{c} = \gamma a^{1/2} \mathbf{D}^{1/2} \mathbf{z}$;

  $\mathbf{w} = \mathbf{x} - \mathbf{y}^{(k)} + \mathbf{M}^{-T}(\mathbf{c} - \mathbf{A}\mathbf{x})$;

  $\mathbf{y}^{(k+1)} = \alpha(\mathbf{y}^{(k)} - \mathbf{y}^{(k-1)} + \tau\mathbf{w}) + \mathbf{y}^{(k-1)}$;

  $\beta = (\theta - \beta\delta)^{-1}$;

  $\alpha = \theta\beta$;

end

Table of Contents

Algorithm structure design

Writing down direct mathmatical expression raises *huge time delay and memory allocation problem*, Even few attemptions for reduction memory(Sparse, using basic matrix inversion operations) were failed.

```
[62]: @benchmark ssor($A, $b, 0.75, maxiter = 10000)

[62]: BenchmarkTools.Trial:
        memory estimate:  859.84 KiB
        allocs estimate:  40012
        --------------
        minimum time:     2.868 s (0.00% GC)
        median time:      2.935 s (0.00% GC)
        mean time:        2.935 s (0.00% GC)
        maximum time:     3.001 s (0.00% GC)
        --------------
        samples:          2
        evals/sample:     1

[61]: @benchmark k3_ssor($A, $b, 0.75, 10000)

[61]: BenchmarkTools.Trial:
        memory estimate:  4.48 GiB
        allocs estimate:  120043
        --------------
        minimum time:     4.434 s (4.42% GC)
        median time:      4.499 s (5.17% GC)
        mean time:        4.499 s (5.17% GC)
        maximum time:     4.565 s (5.89% GC)
        --------------
        samples:          2
        evals/sample:     1
```

```
function sor_temp(A,b,w, max_itter)
    batch = 100
    D = sparse(Diagonal(A))
    M = sparse((1/w-1) * D + LowerTriangular(A))
    N = M - A # automatlly assigned sparse type
    M_inv_N = sparse(M\N)
    M_inv_b = sparse(M\b)
    big_itter = max_itter ÷ batch
    x =zeros(length(b))
    x_pre = zeros(length(b))

    for j = 1 : big_itter
        for i = 1:batch
            x = M_inv_N*x + M_inv_b
        end
        if norm(x - x_pre, 2) < 10^-3
            return x
        end
    end

    return x
end
```

Figure: Compare performance between Julia itersolver and prototype solver

Algorithm structure design

We benchmark Julia iterative solver algorithms and figure out how to reduce computing time and memory allocations.

- The key idea is *DiagonalIndices structure* and sparse based inplacement operation.
- In julia sparse package, SparseMatricCSC type is only storage 3 types of vectors; colum pointers, none zero values and row values
- Iterative solver design Diagonal structure and store one more array ; Diagonal indices of none zero values.

Using Diagonal structure, forward and backward operation can be computed without declaring new Lower or Upper Triangluar matrices.

Algorithm structure design

Two important tips for enhancing performance of iteration solver and sampler :

- Pre-allocation of variables
- writing all operations as in-place computing function

Our algorithms have more advantage in saving memory allocations over 10 times compared with iterative solver and also faster than Julia iteration solver package.

Algorithm structure design

performance of iterative solver and $k3$-iterative solver

```
@benchmark ssor(A, b, 1.6641, maxiter= 64000)
```

```
BenchmarkTools.Trial:
  memory estimate:  3.91 MiB
  allocs estimate:  256010
  --------------
  minimum time:     180.204 ms (0.00% GC)
  median time:      183.230 ms (0.00% GC)
  mean time:        186.453 ms (1.26% GC)
  maximum time:     226.778 ms (20.00% GC)
  --------------
  samples:          27
  evals/sample:     1
```

```
@benchmark sor(A, b, 1.6641, maxiter= 64000)
```

```
BenchmarkTools.Trial:
  memory estimate:  1.96 MiB
  allocs estimate:  128008
  --------------
  minimum time:     86.863 ms (0.00% GC)
  median time:      87.762 ms (0.00% GC)
  mean time:        89.777 ms (1.28% GC)
  maximum time:     133.410 ms (34.79% GC)
  --------------
  samples:          56
  evals/sample:     1
```

```
@benchmark k3_ssor(A, b, 1.6641, 64000)
```

```
BenchmarkTools.Trial:
  memory estimate:  31.64 KiB
  allocs estimate:  27
  --------------
  minimum time:     179.993 ms (0.00% GC)
  median time:      182.355 ms (0.00% GC)
  mean time:        183.060 ms (0.00% GC)
  maximum time:     195.136 ms (0.00% GC)
  --------------
  samples:          28
  evals/sample:     1
```

```
@benchmark k3_sor(A, b, 1.6641, 64000)
```

```
BenchmarkTools.Trial:
  memory estimate:  12.02 KiB
  allocs estimate:  11
  --------------
  minimum time:     85.997 ms (0.00% GC)
  median time:      87.990 ms (0.00% GC)
  mean time:        89.472 ms (0.00% GC)
  maximum time:     100.938 ms (0.00% GC)
  --------------
  samples:          56
  evals/sample:     1
```

Table of Contents

Data matrix: 10 by 10 lattice sparse precision matrix

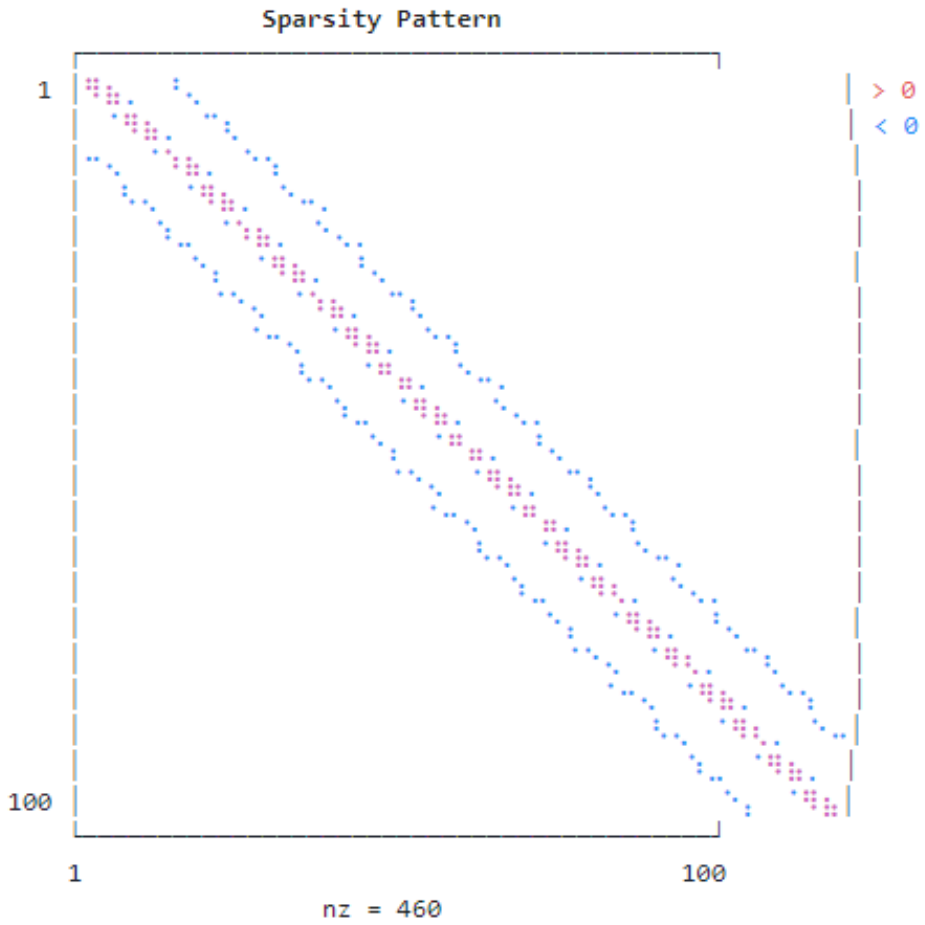

Figure: Sparsity pattern of A. A first order locally linear sparse precision matrix $\mathbf{A}$ considered by Higdon(2006), Rue and Held(2005) is defined by $\mathbf{A}_{ij} = 10^{-4}\delta_{ij} + n_i$ if $i = j$, $\mathbf{A}_{ij} = 10^{-4}\delta_{ij} - 1$ if $i \neq j$ and $\|s_i - s_j\|_2 \leq 1$, $\mathbf{A}_{ij} = 10^{-4}\delta_{ij}$ otherwise.

Table 3: Flops and Iterations of each Solver

Following table is our calculation results for solving $Ax = b$. Note that there are small diffrences comparing to the paper's results but there diffrences are negligible.

Solver	ω	Number of iterations(Paper)	Total Flops(Paper)
Richardson	1	DNC(DNC)	-
Jacobi	-	$6.92(4.01) \times 10^5$	$5.05(5.69) \times 10^8$
Gauss-Seidel	-	$3.20(2.44) \times 10^5$	$1.99(4.34) \times 10^8$
SSOR	1.6641	$6.13(6.7) \times 10^4$	$2.64(2.39) \times 10^6$
SOR	1.9852	1669(1655)	$2.64(2.95) \times 10^6$
Cheby-SSOR	1	1021(958)	$3.37(3.41) \times 10^6$
Cheby-SSOR	1.6641	636(628)	$2.10(2.21) \times 10^6$

Benchmark results for Accelerated solver

We implement a linear iterative-solvers and accelerated solver by Julia. Following image is a result of benchmark on Chebyshev accelerated iterative solver.

```
b_ = copy(b)
@benchmark ssor(A, b_, 1.6641, maxiter=61300)
```

```
BenchmarkTools.Trial:
  memory estimate:  3.74 MiB
  allocs estimate:  245210
  --------------
  minimum time:     152.768 ms (0.00% GC)
  median time:      164.098 ms (0.00% GC)
  mean time:        163.219 ms (0.09% GC)
  maximum time:     172.918 ms (0.52% GC)
  --------------
  samples:          31
  evals/sample:     1
```

```
b_ = copy(b)
@time λ_max,λ_min = eigMm(A, 1.6641)
@benchmark k3_CB_ssor(A, b_, 1.6641, λ_max, λ_min, 640)
```

```
  0.009268 seconds (541 allocations: 752.641 KiB)
BenchmarkTools.Trial:
  memory estimate:  36.89 KiB
  allocs estimate:  33
  --------------
  minimum time:     1.478 ms (0.00% GC)
  median time:      1.716 ms (0.00% GC)
  mean time:        1.735 ms (0.18% GC)
  maximum time:     3.885 ms (58.21% GC)
  --------------
  samples:          2878
  evals/sample:     1
```

Figure: Comparison between SSOR and Cheby-SSOR. Left shows the benchmark results of SSOR implemented in the Julia package IterativeSolvers. Right shows the benchmark results of Cheby-SSOR implemented by us.These shows that our implementation is much faster and memory efficient.

Figure2: Sample Variance error graph

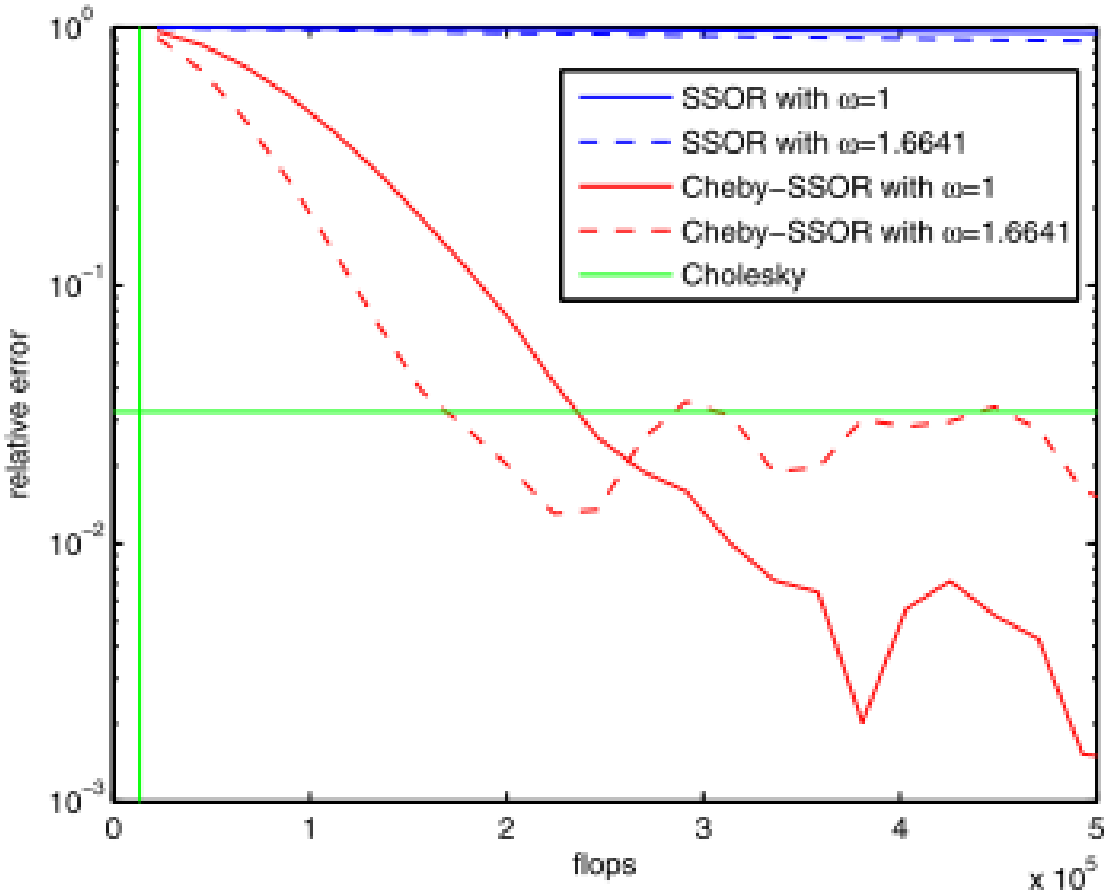

Figure: Relative error in covariance $\|\mathbf{A}^{-1} - \mathbf{S}_y^{(k)}\|_2 / \|\mathbf{A}^{-1}\|_2$ versus number of iterations for a sampler implemented with SSOR and $\omega = 1$, SSOR with optimal relaxation $\omega = 1.6641$, and SSOR with Chebyshev acceleration.

Figure2: Sample Variance error graph

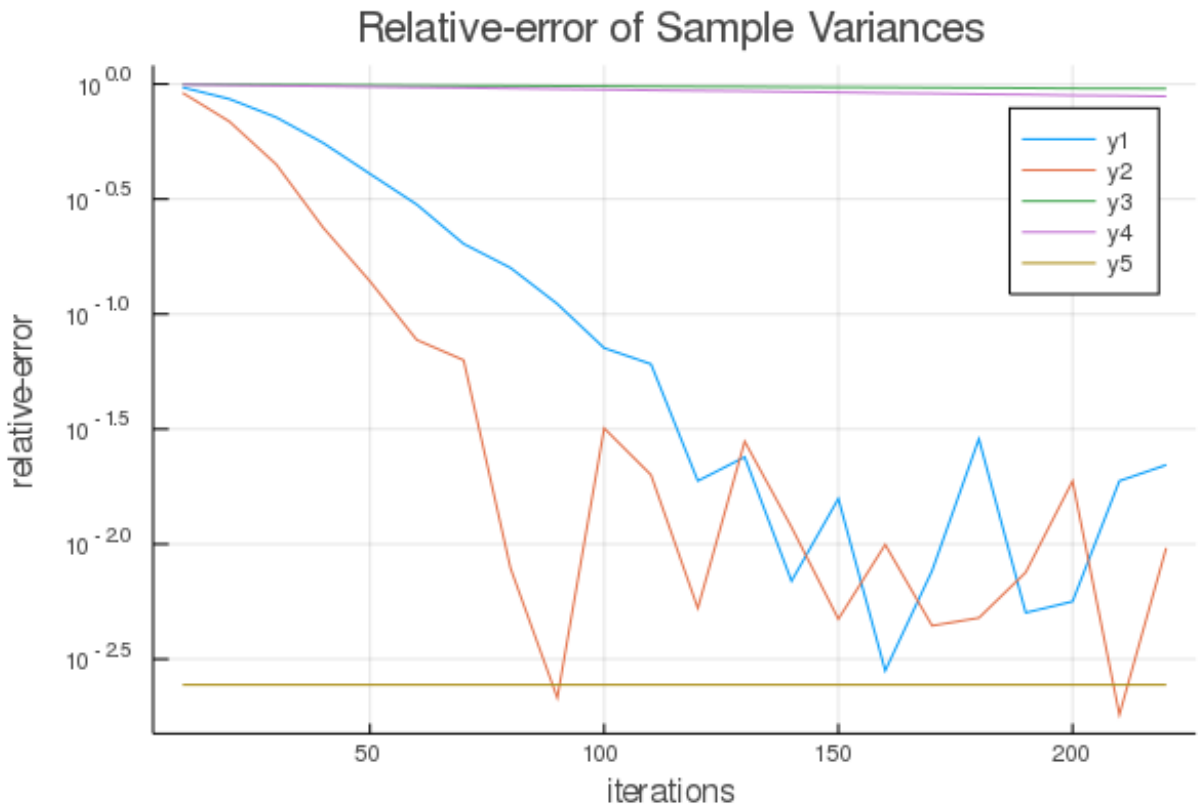

Figure: y_1 denotes Cheby-SSOR with w $= 1$ and y_2 denotes Cheby-SSOR with w $=$ 1.6441. y_3 is SSOR with w $= 1$ and y_4 is SSOR with w $= 1.6441$. Finally, y_5 is non-iterative cholesky sampler of algorithm 1 in the paper.

Table of Contents

Biofilm image data

A.Parker et al.(2018, JASA) used a Accelerated Samplers for recovering surface of biofilm data generated by confocal microscpoes(CM).

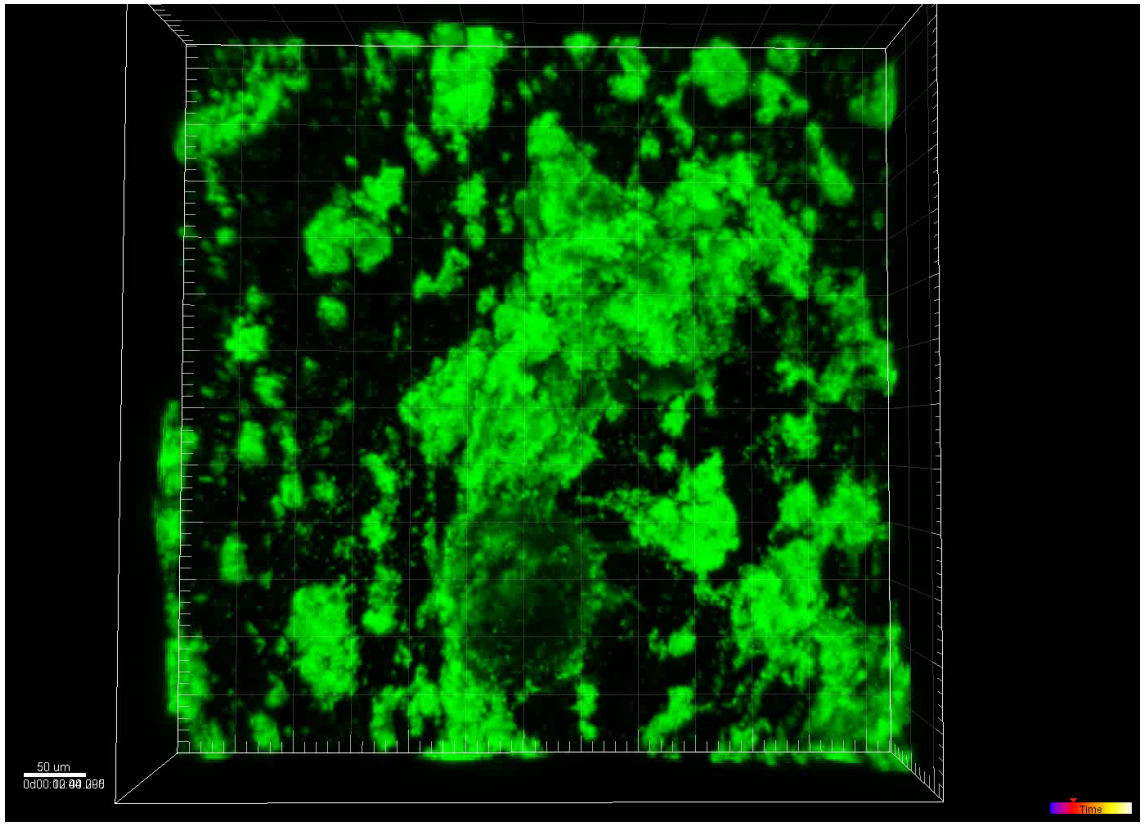

Figure: The paper uses a sequence of CM images over time of a green fluorescing *Staphylococcus aureus* biofilm grown under controlled conditions. Above picture is a first frame of video of CM images. Each CM image consists of $620\mu m$ by $620\mu m$ by $112\mu m$.

Modeling: Gaussian Model for Biofilm surface

- Note that we can consider a height information as a discretize observed surface representation. We want to recover the whole surface information θ.
- Given a surface representation, $y \in \mathbb{R}^{512\times 512}$, the linear statistical model that we apply to the surface profile is

$$y = F\theta + \epsilon.$$

- The likelihood is $\pi(y|\theta, \Sigma_y) = N(F\theta, \Sigma_y)$, as $\epsilon \sim N(0, \Sigma_y)$. The prior is assumed as $\pi(\theta) = N(0, \frac{1}{\lambda} W^{-1})$ with λ for smoothing, W, the Laplacian $512^2 \times 512^2$ precision matrix (extension of 6.1).
- Using Bayesian approach, our goal is to estimate the posterior, $\pi(\theta, \sigma^2, \lambda|y)$, and especially

$$\pi(\theta|y, \sigma^2, \lambda) = N(\frac{1}{\sigma^2} A^{-1} y, A^{-1})$$

with precision matrix $A = \frac{1}{\sigma^2} I + \lambda W, \quad 1/\sigma^2, \lambda \sim Gamma(\alpha = 1, \beta = 10^{-4})$.

Required studies for large Real data analysis

Solving linear inverse problem for large matrices ($512^2 \times 512^2$ matrix for our problem) using Cheby-accelerated sampler requires to obtain an extreme eigen values of $M_{SSOR}^{-1}A$. In fact, this problem should be seen in the authors' series of research in the long term.

- Fox, C., and Parker, A.(2012) Introduced a conjugate gradient sampler which then could inexpensively estimate some of the eigenvalues of A.
- Fox, C., and Parker, A.(2014) First suggested a Cheby-accelerated sampler.
- Fox, C., and Parker, A.(2017) Proved convergence in distribution for Gibbs samplers corresponding to *any* matrix splitting and accelerated by *any* polynomial that is independent of the Gibbs iteration.
- Parker, A. et al.(2018) Combined all this methods and proposed PCG and *PCG-Chebyshev Accelerated Sampler*.

What we implemented

The extreme eigenvalues of the Lanczos matrix, found at a negligible k^2 flops when $k << n$, are the required extreme eigenvalues of $M^{-1}A$.

Algorithm 2: PCG-Chebyshev Accelerated Sampler of $N(A^{-1}b, A^{-1})$

input : SPD precision matrix A, $M = M_1 M_1^T$ where M_1 is a matrix splitting of A, initial state θ^0, b, and initial estimate of x^0 of $A^{-1}b$, PCG residual stopping criterion ϵ_{PCG}, maximum number of PCG iterations k_{PCG}, number of Chebyshev iterations k_{Cheby}

output: $\theta \dot{\sim} N(A^{-1}b, A^{-1})$ and $x \approx A^{-1}b$

PCG sampling

  input : θ^0, x^0, A, split preconditioner $C = M_1$, $\epsilon = \epsilon_{\text{PCG}}$, $k_{\max} = k_{\text{PCG}}$

  output: $\theta_{\text{PCG}} \dot{\sim} N(0, A^{-1})$, $x_{\text{PCG}} \approx A^{-1}b$ and $\{\gamma_k, \beta_k\}$

  Implement Algorithm 1, get approximate solution x^{k+1} and approximate sample θ^{k+1};

end

Get the extreme eigenvalues of $M^{-1}A$ from $\{\gamma_k, \beta_k\}$ using the prescription in (Parker and Fox (2012), Lemma 2.1);

Chebyshev sampling

  input : Number of sampler iterations k_{Cheby}, $\theta^0 = \theta_{\text{PCG}}$, $x^0 = \theta_{\text{PCG}}$, $b_{\text{Cheby}} = 0$, extreme eigenvalues of $M_{-1}A$

  output: $\theta_{\text{Cheby}} \dot{\sim} N(0, A^{-1})$

  Run Algorithm 3 of Fox and Parker (2014) for k_{Cheby} iterations. At the k_{Cheby}th iteration, get approximate sample $\theta^{k+1}_{\text{Cheby}}$;

end

$\theta = \theta_{\text{Cheby}} + x_{\text{PCG}}$ and $x = x_{\text{PCG}}$;

Figure: PCG-Accelerated Sampler from Algorithm 2 of (2018, JASA)

What we implemented

- Fox, C., and Parker, A.(2012) PCG Accelerated Sampler
- Fox, C., and Parker, A.(2014) Cheby-accelerated sampler
- Fox, C., and Parker, A.(2017) Cheby-accelerated sampler
- Parker, A. et al.(2018) PCG-Chebyshev Accelerated Sampler
- Additional (Sparse) Lanczos matrix, Generalized lattice sparse precision matrix

Result

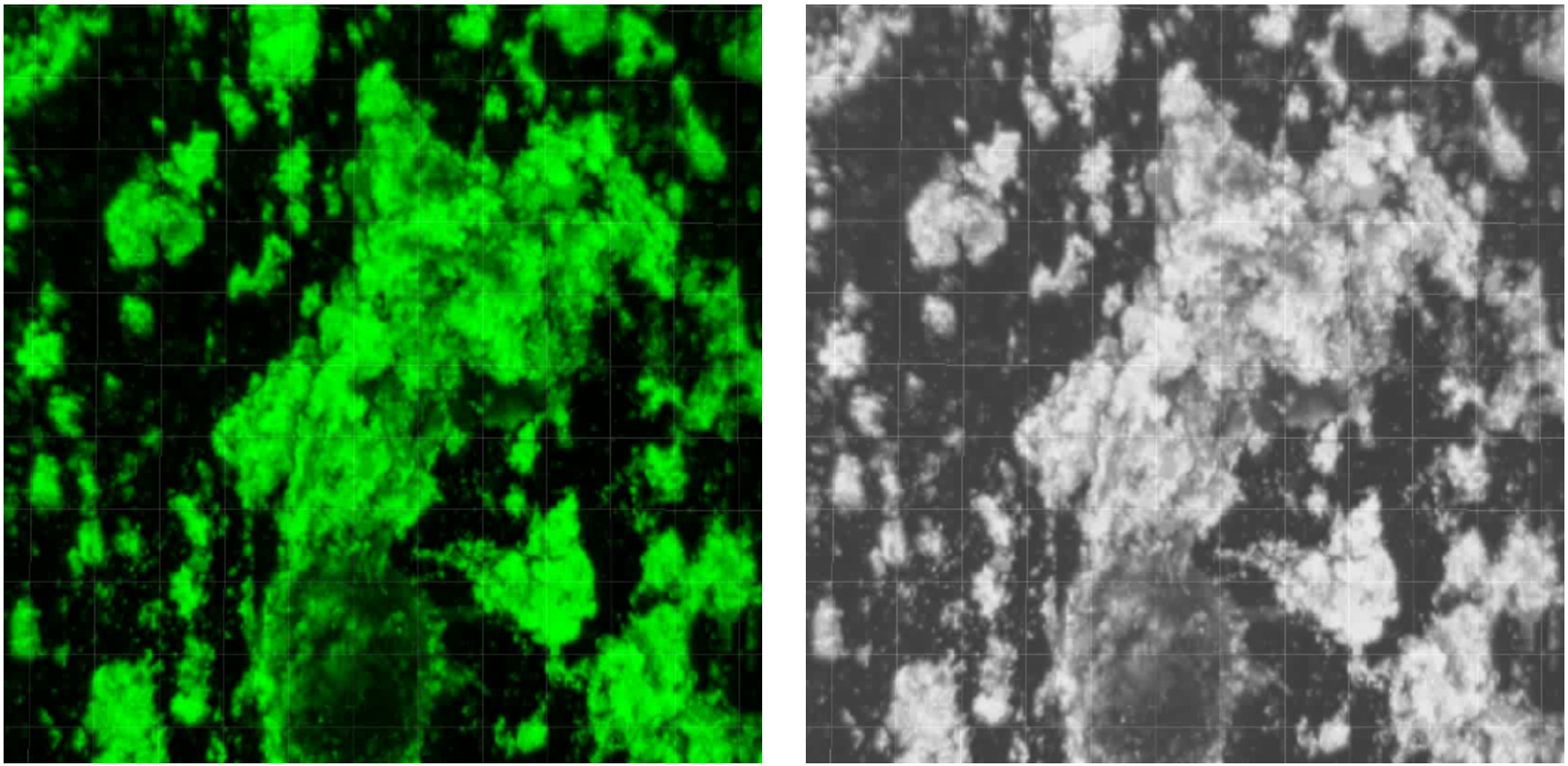

Figure: Observed image from the video for frame 1 and the PCG-accelerated sampled image

Thank You