An Implementation of the Markov Chain Monte Carlo Technique

Alistair Boyle
Biomedical Engineering
Carleton University
Ottawa, Ontario, Canada
alistair.js.boyle@gmail.com

April 14, 2009

Abstract

Markov Chain Monte Carlo was used in the solution of regularized Computed Tomography back projection. The effect of reducing the information used in reconstruction was investigated and regions of increased variance around some types of image artifacts were observed.

1 Introduction

This report summarizes the investigation of a technique called "Markov Chain Monte Carlo" that is employed to explore the posterior distribution of two-dimensional Computed Tomography (CT) images.

1.1 Computed Tomography

Computed Tomography (CT) images are resolved using one of a number of numerical techniques collectively referred to as tomosynthesis.

In a CT scan, an image of a target is reconstructed by emitting x-rays from a source and collecting corresponding measurements on the other side of the target. In first-generation CT scanners, these x-ray particles are emitted in a parallel plane by the x-ray source.[1] As an x-ray travels through the target, some of the energy of the x-ray is absorbed by the target. Denser regions absorb greater amounts of energy, such that a dense region within the target has a relative reduction in the measured energy behind that denser region. (Figure 1)

An image of the internal structure of the target is reconstructed from a set of measurements. A large set of measurements may be required to obtain enough information to reconstruct an accurate image. The necessary information is obtained by taking many measurements from slightly different angles around the circumference of the target.

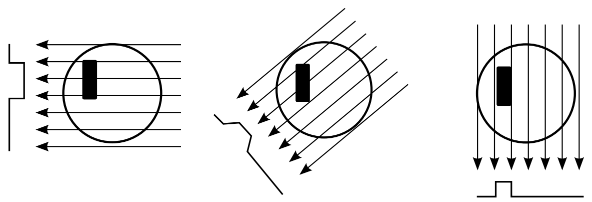


Figure 1: X-ray projections through a target; the measurements change as the angle of the emitted x-rays changes.

The process of obtaining measurements can be modelled mathematically using the Radon transform. Let $(x(t), y(t))$ be a straight line, a single x-ray particle travelling through the target. The line can be defined in terms of the perpendicular distance from the origin s and the angle from the x-axis α along the line t through A–A'. (Figure 2)

$$(x(t), y(t)) = t(\sin\alpha, -\cos\alpha) + s(\cos\alpha, \sin\alpha) \tag{1}$$

Using this definition of a line, the Radon transform is the line integral:

$$\begin{aligned} \mathbb{R}[f](\alpha, s) &= \int_{-\infty}^{\infty} f(x(t), y(t))dt \qquad (2) \\ &= \int_{-\infty}^{\infty} f(t(\sin\alpha, -\cos\alpha) + s(\cos\alpha, \sin\alpha))dt \qquad (3) \end{aligned}$$

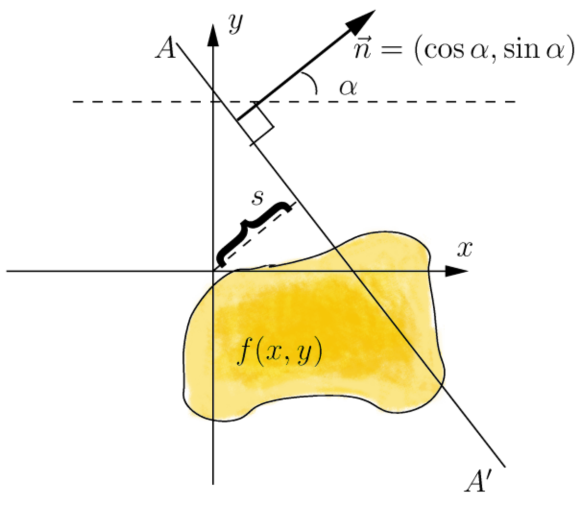


Figure 2: Radon transform variables[6]: a line along t through A–A' can be defined in terms of the angle from the x-axis α and perpendicular distance from the origin s. The line t cuts through a region that's density is defined by $f(x, y)$.

To find the image, given the measurements (the inverse solution), the Fourier Slice Theorem can be employed. It can be shown that, with an infinite number of measurements, a completely accurate reconstruction can be built.[6] Unfortunately, this is not particularly helpful for practical reconstructions since we are limited to a finite number of measurements. In practical CT systems, the number of measurements that can be taken is further restricted by physical limits to the range of angles from which measurements can be taken (e.g. dental x-rays) and x-ray exposure limits for the target.

1.2 Discrete Projection

The Radon transform was discretized by means of an interpolatory linear projection matrix for a given angle α. This projection matrix takes an image and transforms it into measurements consisting of discrete values on a line perpendicular to the direction of the x-rays.

The interpolatory linear projection matrix is derived by first taking the rotation of $[x\ y]^{\mathrm{T}}$ onto the new basis $[t\ s]^{\mathrm{T}}$.

$$\begin{bmatrix} t \\ s \end{bmatrix} = R(\alpha) \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sin(\alpha) & \cos(\alpha) \\ \cos(\alpha) & -\sin(\alpha) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4}$$

The rotation is applied to a set of coordinates representing rays through the image and a summation along each ray t gives the density of the image pixels projected onto the target for a given angle α. Rays that pass through the square pixels of the original density are interpolated when the ray does not pass through the exact centre of the pixel such that a single pixel of the original image may be assigned to two or three pixels in the projection. (See `makeproj()`, Appendix A.2.)

The resulting matrix is called the projection matrix $P(\alpha)$. A series of these projection matrices with differing angles α are concatenated to calculate a set of projections b. This matrix describes the forward solution for projection and is referred to as the sensitivity matrix H.

$$H = \begin{bmatrix} P(\alpha_1) \\ P(\alpha_2) \\ \vdots \end{bmatrix} \tag{5}$$

The inverse problem, that of finding the image u from the measurements b can be solved such that

$$Hu = b \qquad \rightarrow \qquad u = H^{\dagger} b \tag{6}$$

Solving the inverse problem corresponds to minimizing the 2-norm residual $\arg\min\{||Hu - b||_2^2\}$.

An analysis of the sensitivity matrix H by Singular Value Decomposition (SVD) shows the problem to be rank-deficient since the singular values show a significant gap.(Figure 3) This indicates that small changes in the measurements will be magnified into large changes in the image. Indeed, adding a small amount of Gaussian white noise to the measurements

(1%) results in an image with no recognizable features. The ill-posed nature of the problem is exacerbated by reducing the angles and/or the number of projections which reduces the amount of information on which the reconstruction can be based.

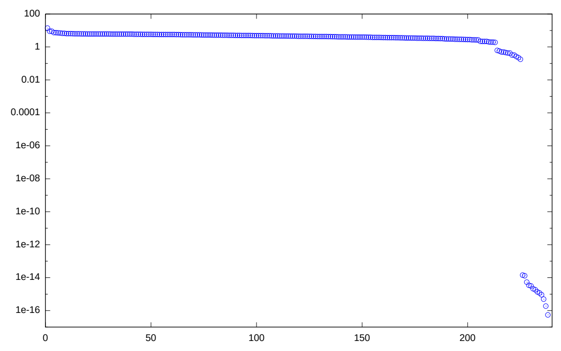


Figure 3: Singular values for the sensitivity matrix with 6 projections, 30° apart. Vertical-axis: singular value magnitude, horizontal-axis: nth singular value. A significant gap in the singular values between the 231st and 232nd leaves 15 very small singular values which, when H is inverted, significantly magnify the effect any noise in the measurements.

2 Traditional Regularization

Solving the inverse problem is a challenge due to the ill-posed nature of the sensitivity matrix H. To arrive at a solution that converges to a reasonable estimate of the true image, additional information must be added to the system of equations. This can be done by adding "regularization" to the inverse problem.

Various criteria can be used in attempting to restrict the possible solution set of the inverse problem, such as the following:

- minimizing the residual norm while constraining the solution's values,
$$\arg\min\{\ ||Hu - b||_2^2\ \}\ ,\quad u \in S \tag{7}$$
- minimizing the residual norm while constraining the solution's magnitude,
$$\arg\min\{\ ||Hu - b||_2^2\ \}\ ,\quad L(u) < \delta \tag{8}$$
where L is a function that is a measure of the magnitude of u, referred to as a "smoothing norm";
- minimizing the magnitude while constraining the residual norm,
$$\arg\min\{\ ||L(u)||_2^2\ \}\ ,\quad ||Hu - b||_2^2 < \delta \tag{9}$$
and

- minimizing the residual norm and the solution magnitude together (Tikhonov Regularization),

$$\arg\min\{\ ||Hu - b||_2^2\ +\ \lambda^2||L(u - u_0)||_2^2\ \} \tag{10}$$

  where λ is a hyper-parameter that determines how much regularization to apply and u_0 represents *a prior* knowledge of the expected solution.

Some possible choices for the discrete smoothing norm L are as follows:

- the identity matrix I, representing a penalty on the total magnitude of the solution,
- a diagonal set of penalty weights $diag(w)$, which allows the application of non-uniform penalties to the magnitude of the solution in different regions, and
- discrete approximations of derivatives, the first-order derivative L_1, second-order derivative L_2, and so on.

This report focuses on Tikhonov regularization where the identity matrix was selected as an initial discrete smoothing norm. This proved to be satisfactory for regularizing the image enough to converge to a solution, but the image contained a significant amount of noise. A second choice of smoothing norm was a one-dimensional first-derivative smoothing norm which succeeded in reducing noise in that dimension. Finally, a discrete smoothing norm that approximates a first-order derivative in two-dimensions was selected. This significantly reduced the noise in the image.

A zero prior was selected ($u_0 = 0$) since no *a priori* structural knowledge of the original image was assumed.

The hyper-parameter was initially selected through trial-and-error and was followed by selection through application of the L-curve approach which chooses a hyper-parameter value that is an optimal trade-off between the residual norm and the smoothing norm magnitudes. [2] (Figure 4)

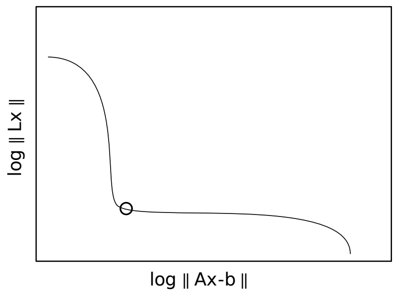


Figure 4: L-curve; *o* marks the optimal hyper-parameter value; on the x-axis, moving too far to the left represents under-regularizing the solution while moving too far to the right represents over-regularizing the solution.

3 Statistical Regularization

A statistical approach to the inverse problem recognizes that all variables in the problem can be modelled as random variables with a probability distribution. The distribution of a variable reflects the degree of certainty in that variable's true value. The random variable is represented by its uppercase counter-part such that $u \mapsto U$ and $b \mapsto B$, and the probability of that random variable having a specific value is $P(u)$ or $P(b)$.

From a Bayesian perspective, the data about the problem can be classified into three types: prior or marginal probabilities, conditional probabilities, and posterior probabilities. The *a priori* information is represented as a marginal distribution $P(b)$, while the forward model, the probability of the output given the input, is represented as a conditional probability $P(b|u)$. The posterior distribution $P(u|b)$, the likelihood of the input given the output, is given by Bayes formula

$$P(u|b) = \frac{P(b,u)}{P(b)} = \frac{P(b|u)P(u)}{P(b)} \tag{11}$$

The prior distribution of the measurements $P(b)$ acts as a uniform normalizing constant and can therefore be ignored for most numerical solution techniques.

If the random variables are viewed as having a Gaussian distribution, then the bias and variance of those variables has an intuitive meaning corresponding to the mean error and uncertainty respectively.

Viewing the solution in terms of its bias and variance offers a number of insights into the solution. First, it clearly indicates how much confidence should be placed in the solution. A large variance indicates little confidence in the variable's value. Second, when a "true" value is known, it helps to indicate the bias in a variable and can thus help to identify trade-offs in bias and variance of a technique. Third, it places emphasis on choosing a prior, where not choosing a prior is the same as selecting a zero prior.

Traditional regularization techniques obtain a single estimate of the solution, in many cases by removing the ill-posed nature of the solution in a somewhat ad-hoc manner, asking "what is the value of this variable?" The statistical approach turns this question around to instead ask "what is our information about this variable?" [3]

A couple of techniques for obtaining an estimate of the solution are the Maximum *A Posteriori* (MAP) estimate, the Conditional Mean (CM), and the Maximum Likelihood (ML).

$$x_{\mathrm{MAP}} = \arg\max\{\ P(u|b)\ \} \tag{12}$$

$$x_{\mathrm{CM}} = E\{u|b\} = \int uP(u|b)du \tag{13}$$

$$x_{\mathrm{ML}} = \arg\max\{\ P(b|x)\ \} \tag{14}$$

For the MAP estimate, if the distribution of the conditional probability $P(u|b)$ is Gaussian, then the maximum of the distribution is the mean. For the ML estimate, finding the maximum often corresponds to solving the non-regularized inverse problem and, in ill-posed or rank-deficient inverse problems, is not likely to converge.

Unfortunately, in many cases the conditional probability $P(b|x)$ is not available or is difficult to obtain which makes the use of Bayes formula (11) infeasible. One solution to this problem is to explore the posterior distribution more directly through a technique such as Markov Chain Monte Carlo.

4 Markov Chain Monte Carlo

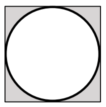
Markov Chain Monte Carlo (MCMC) refers to a combination of two statistical techniques: Markov Chain probabilities and the Monte Carlo method. The two basic algorithms for implementing MCMC are the Metropolis-Hastings and the Gibbs Sampling algorithms.

4.1 Monte Carlo

The Monte Carlo method refers to a method where random, uncorrelated samples are drawn with replacement. This sampling technique is applied in sampling from a posterior distribution, integration in high-dimensional spaces, simulated annealing, and learning algorithms.

The general steps in the Monte Carlo method are to define the range of the inputs, draw a set of samples with replacement, perform some calculation on each sample, and find some information about the ensemble of the calculations.

One illustrative example of the Monte Carlo technique is in building an estimate of π. To start, draw a square, and then draw a circle inside the square that touches the edges of the square. Throw a set of markers into the square (the randomly drawn samples), and determine for each marker if it is within the circle. The fraction of the markers inside the circle is approximately $\pi/4$. [7]



In general, Monte Carlo methods will converge by the "law of large numbers": [7]

$$P\left(\lim_{N\to\infty}\frac{1}{N}\sum_{i=0}^{N}x_i=\mu\right)=1 \tag{15}$$

The Monte Carlo method can be improved in some scenarios by getting the algorithm to sample in regions that are more significant.

4.2 Markov Chain

A Markov Chain is "a mathematical model for stochastic systems whose states, ... are governed by a transition probability. The current state in a Markov chain only depends on the most recent previous states." [9] A first-order Markov Chain only depends on the previous state, while an m-th order Markov Chain depends on the last m states.

Combining the two concepts, the simplest form of Markov Chain Monte Carlo is a "random walk" where the last sample is used as the basis for deciding the next "step" based on a distribution, such as a normal distribution with a given variance σ^2 and a mean centred at the current location. This concept is the basis for the Metropolis-Hastings algorithm.

4.3 Metropolis-Hastings

The Metropolis-Hastings algorithm can approximate any distribution $P(x)$ so long as a reasonable estimate of the distribution can be calculated for a given location x. For each iteration of the algorithm, a new location is chosen from a proposal density $x' \sim Q(x'; x_t)$ where Q is a density based on x_t. The new location is accepted based on an acceptance ratio:

$$a = \frac{P(x')Q(x'; x_t)}{P(x_t)Q(x_t; x')} \tag{16}$$

where a is the likelihood of accepting the new location. If $a \geq 1$, the new location x' is accepted. If the proposal density Q is symmetric, then only the relative probabilities of the old and new location affect the outcome. If accepted, the current location is updated $x_{t+1} = x'$, then the algorithm repeats.

An example of a simple proposal density is the normal distribution

$$Q(x'; x_t) \sim \mathcal{N}(x_t, \sigma^2 I) \tag{17}$$

such that a move with the variance σ^2 from x_t will be proposed.

If starting at a random location, a period of iterations will pass while the algorithm moves to the significant regions of the distribution. This period is referred to as a "burn-in" period. The burn-in period is dependant on the variance of the proposal density since a small variance in the proposal density will result in small steps for the random walk, but a large variance may fail to explore the probability of interest sufficiently and will also result in a low acceptance rate since the probability of the new location $P(x')$ is likely to be small.

A generalization of the Metropolis-Hastings algorithm that removes the lost work due to rejected proposals is the Gibbs Sampler.

4.4 Gibbs Sampling

A Gibbs Sampler can be used to approximate the joint probability distribution $P(b, u)$ which in turn approximates Bayes formula (11)

$$P(u|b) \propto P(b, u)$$

The Gibbs sampler approximates the joint probability by taking the conditional probability of each variable based on all other variables. This is the definition of a Markov Chain with a stationary distribution approximating the joint probability distribution. A Gibbs sampler requires at least two dependant variables to alternate between.[8]

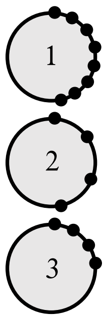
The main advantage of Gibbs sampling is that it's easier to sample from the conditional probabilities rather than integrate over the multiple variables of a joint probability distribution (a multidimensional integration problem).

The Gibbs sampler can be seen as a generalization of the Metropolis-Hastings algorithm in the sense that it is proposing moves to new locations using kernels Q that are the conditional probabilities, and the proposal is always accepted. The Gibbs sampler converges not because of the acceptance of a proposal but because the proposed moves are directly related to the conditional probabilities.

While both Gibbs Samplers and the Metropolis-Hastings algorithms do converge, convergence can be very slow in instances where there are regions of low probability separating regions of high probability since the algorithms traverse the probability distribution. Getting from one region to the other can take a large number of tries. A second failure mode occurs when the likelihood of a particular outcome significantly outweighs other outcomes. The algorithm can return long sequences relating to a particular outcome without providing information on the other outcomes such that an unreasonable number of samples must be taken to obtain an approximation of the true distribution. Both of these issues imply that a very large number of samples may be required to obtain a good estimate of the true posterior distribution.

5 Results

An original image containing two rectangular targets was simulated using a forward model that projects the two-dimensional image onto a one-dimensional line thus simulating the x-ray process. Gaussian white noise was added to each projection at 10% of the mean amplitude of all projections.



Three scenarios were simulated:

1. a full angle (0° – 180°) set of projections with many (60) evenly spaced projections,
2. a full angle (0° – 180°) set of projections with very limited (10) evenly spaced projections, and
3. a limited angle (0° – 90°) set of projections with very limited (10) projections.

The resulting projections were interpolated at a 2:1 ratio to avoid an inverse crime where the forward and inverse model discretizations exactly match giving unreasonably optimistic reconstruction results. The forward model was computed on an 80x80 pixel image while the reconstructions were built on a 40x40 image.

Each image was reconstructed using a discrete smoothing norm that approximates a first-order derivative in two dimensions. A Gibbs Sampler was employed to take 500 samples of the image, where the hyper-parameter and image values were used as variables. For each iteration, the pixels of the image were treated as independent Gaussian distributed variables and were thus drawn as a set. The image prior u_0 was zero, but noise was added which implied a degree of uncertainty in the prior.

$$\begin{aligned} P(u|\lambda,b) &\propto \exp\left(-\frac{1}{2}\frac{\lambda^2}{{\sigma_p}^2}||L(u-u_0)||^2 - \frac{1}{2{\sigma_b}^2}||b-Hu||^2\right) && (18)\\ \eta &\sim \mathcal{N}(0,\sigma_b) && (19)\\ \zeta &\sim \mathcal{N}(0,\sigma_p) && (20)\\ u &= \begin{bmatrix} {\sigma_b}^{-1}H \\ \lambda{\sigma_p}^{-1}L \end{bmatrix}^\dagger \begin{bmatrix} {\sigma_b}^{-1}b+\eta \\ \lambda{\sigma_p}^{-1}Lu_0+\zeta \end{bmatrix} && (21) \end{aligned}$$

where the variance of the projection noise variance ${\sigma_b}^2$ was assumed to be known exactly (the variance introduced in the forward stage of the problem), and the prior noise variance ${\sigma_p}^2$ was 10% of the projection noise variance.[3]

For each iteration, the conditional probability of the hyper-parameter λ was calculated based on a Rayleigh model of the hyper-parameter (since the hyper-parameter should have a positive value) conditioned on the most recent image u.

The hyper-parameter is assigned a Rayleigh distribution since it must be a positive value.

$$P(\lambda) = \frac{\lambda^2}{{\lambda_0}^2}\exp\left(-\frac{1}{2}\left(\frac{\lambda^2}{{\lambda_0}^2}\right)^2\right), \quad \lambda > 0 \qquad (22)$$

and the conditional probability of the image based on the hyper-parameter is

$$P(u|\lambda) = \frac{\lambda^n}{\sigma_p\sqrt{(2\pi)^n}}\exp\left(-\frac{1}{2}\frac{\lambda^2}{{\sigma_p}^2}||L(u-u_0)||^2\right) \qquad (23)$$

where n is the dimensionality of the data, in this case $n = 40 \times 40 = 1600$.

The conditional probability of the hyper-parameter can be found: by using Bayes rule (11), by assuming that the image is conditionally independent of the hyper-parameter, and by assuming that the measurements are independent of the hyper-parameter.

$$\begin{aligned} P(u,b|\lambda) &= P(u|\lambda)P(b|\lambda), \text{ if cond. indep.} && (24)\\ P(b) &= P(b|\lambda) && (25)\\ P(u,b) &= P(u|b)P(b), \text{ by Bayes} && (26)\\ \text{then } P(\lambda|u,b) &= \frac{P(u,b|\lambda)P(\lambda)}{P(u,b)} && (27)\\ &= \frac{P(u|\lambda)P(b|\lambda)P(\lambda)}{P(u,b)} && (28)\\ &= \frac{P(u|\lambda)P(b)P(\lambda)}{P(u|b)P(b)} && (29)\\ &= \frac{P(u|\lambda)P(\lambda)}{P(u|b)} && (30) \end{aligned}$$

Recognizing that $P(u|b)$ is a normalizing constant

$$\begin{aligned} P(\lambda|u,b) &\propto P(u|\lambda)P(\lambda) && (31)\\ &\propto \lambda^{(n+2)}\exp\left(-\frac{1}{2}\frac{\lambda^2}{{\sigma_p}^2}||L(u-u_0)||^2 - \frac{1}{2}\left(\frac{\lambda^2}{{\lambda_0}^2}\right)^2\right) && (32) \end{aligned}$$

From this distribution, a new hyper-parameter λ was drawn $\lambda \sim P(\lambda|u, b)$. [3]

For each iteration, the sensitivity matrix H was also recalculated with normally distributed projection angles with a variance of 1°, simulating modelling errors in the system.

The starting point for the Gibbs sampler was taken to be an estimate of the image

$$u = \begin{bmatrix} H \\ \lambda L \end{bmatrix}^\dagger \begin{bmatrix} b \\ \lambda L u_0 \end{bmatrix} \tag{33}$$

and a conservative prior for the hyper-parameter $\lambda_0 = 10$ was selected. By starting with these values, rather than a random image and a smaller hyper-parameter, the burn-in period is removed or significantly reduced since the algorithm is starting somewhere within the desired distribution.

The image u and hyper-parameter λ were saved for each iteration.

A histogram of the hyper-parameter was also plotted which illustrated how the hyper-parameter distribution for each scenario differs. The hyper-parameter is plotted $\log_{10}$ with the mode of the distribution indicated beneath the plot.

For each scenario, the image titled "MAP" is the mean of the images. The variance image is normalized against the MAP image and plotted $\log_{10}$ because of the large variance in some pixels. The maximum and minimum values shown under the variance image are $\log_{10}$ normalized.

The choice of method in plotting the variance is somewhat subjective, and it should be recognized that the regions where the mean value is high naturally appear as regions of reduced variance in the corresponding plot. Of particular note, are the regions which show large variance and mean since these regions represent significant regions of uncertainty.

The outcome for the three scenarios can be seen in Figure 5, Figure 6, and Figure 7. (See Appendix A.1 for code used to generate these figures.)

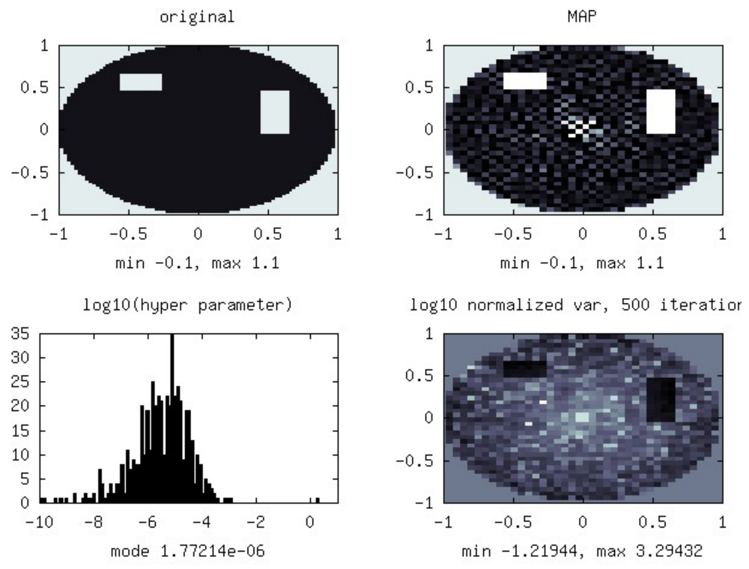


Figure 5: Results for 500 iterations, 60 projection angles over 180°

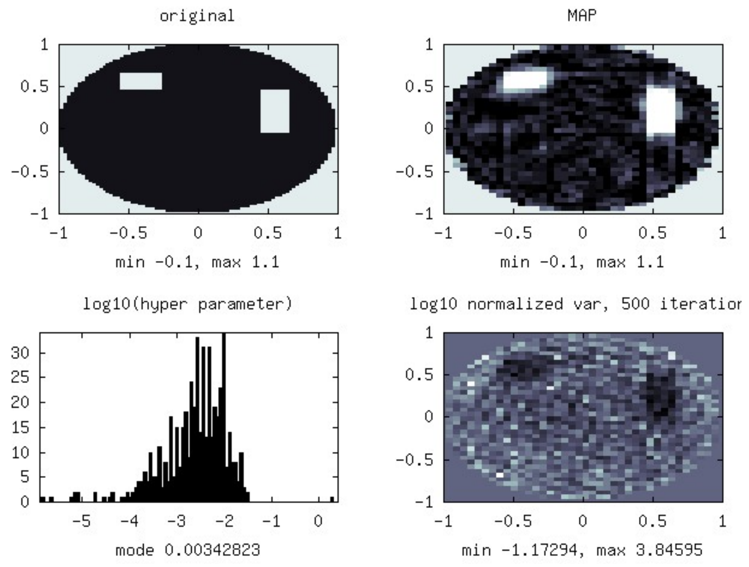


Figure 6: Results for 500 iterations, 10 projection angles over 180°

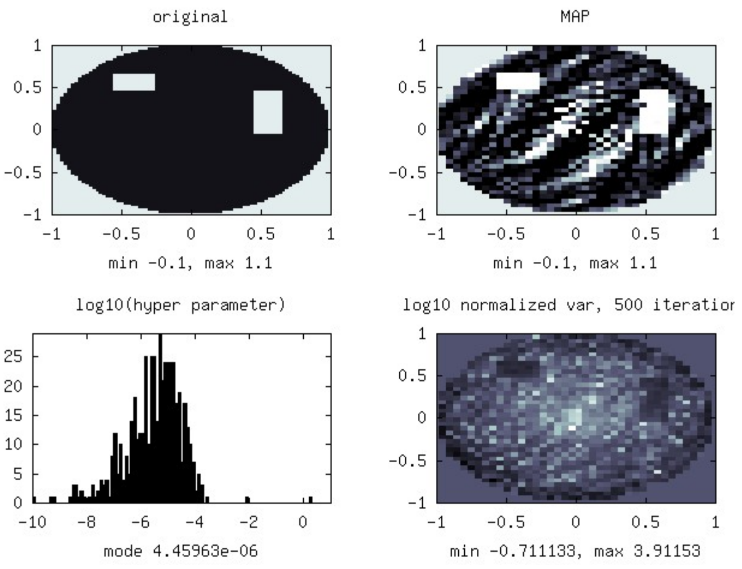


Figure 7: Results for 500 iterations, 10 projection angles over 90°

6 Discussion

In Figure 5, the reconstructed image showed sharp edges around the target objects and limited variance except near the centre of the image where some noise appears. The mode of the hyper-parameters was found to be 1.77×10^{-6}. The noise speckling exhibited in the image is particularly strange since it seems to occur on alternating pixels.

In Figure 6, the reconstructed image showed significantly blurred edges around the target objects and much greater variance across the entire image. The actual range of the variance is quite similar to Figure 5. The variance near the corners of the targets seems to be quite high in some cases which indicates uncertainty about those corners. The mode of the hyper-parameters was found to be 3.44×10^{-3} which indicates significantly more smoothing was applied when compared to Figure 5.

In Figure 7, the reconstructed image showed surprisingly sharp edges around the target objects but also an interesting swirl pattern through the image. This swirl in the image is likely due to the lack of information in the range of projection angles available to reconstruct the image. The variance over the swirls indicates that there is a large amount of uncertainty about them, particularly in contrast to the regions that are identified as targets in the original image. This contrast is particularly apparent when considering that the target objects and the swirls near their maxima appear to be near the same magnitude which indicates that the normalization applied to the variance should be equivalent. The mode of the hyper-parameters was found to be 4.46×10^{-6} which indicates smoothing similar to Figure 5.

6.1 Projection Model Errors

The code for projecting the pixels of the image is one source of potential errors in the system. The current implementation uses a square pixel and projects the pixel onto a line. Because of the square shape of the pixel, projections at angles that are not 90° will result in variation in the spread of the pixels onto the projection. The situation will be worst at 45° where the pixels of the image will be spread across 3 pixels of the projection since the diagonal of the square is wider than the length and breadth.

A solution to this problem would be to treat the pixels as circular. An image pixel will therefore be split into at most two projection pixels. A further advantage of this technique is that the area of the pixel allocated to each portion of the projection is solely a function of the distance from the centre of the image pixel since the circular pixels are now radially symmetric. Therefore, the calculation of how much of each pixel falls into each projection 'bin" is a matter of finding how far each ray falls from the centre of each pixel and then calculating the area fraction for each pixel.

The bi-axial behaviour of the current implementation of the projection approximation may explain some of the speckling seen most clearly in Figure 5. A plot of the fractional area versus distance from the centre of a circle at which a ray splits the circular pixel (Figure 8) shows that a linear approximation of the circular pixel may be good enough to remove most of the artifacts. (See Appendix A.3 for code.)

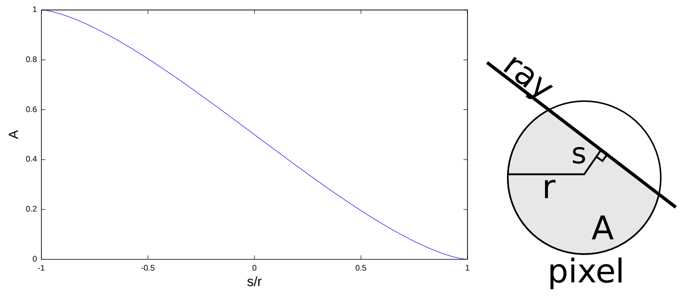


Figure 8: Fractional area of a circle A split by a ray s/r from the centre

6.2 Discrete Smoothing Norm

A modification that might have made a significant difference in the effectiveness of the algorithm is the selection of discrete smoothing norm. The original image has sudden jumps in magnitude when crossing the image, and these changes are not handled well by a discrete smoothing norm such as the first-derivative approximation. This smoothing norm will try to smooth out those edges as can be seen in Figure 6 where the hyper-parameter has been increased relative to Figure 5.

A better choice for discrete smoothing norm, provided the nature of this image was known ahead of time, might be the Total Variation (TV) smoothing norm. TV applies the 1-norm, instead of the 2-norm, to the inverse problem and thus penalizes the total variation in the image rather than the jumps near object boundaries.

6.3 Run Times

Run times for the computation on the images was found to be a limitation in a number of ways. It was found through trial-and-error that the solution time for the inverse matrices grew significantly if the images were made much larger than 40×40 pixels. At this size, calculations took between approximately 5 and 10 seconds, depending on the number of angles used in the reconstruction.

The MCMC algorithm, assuming no burn-in period, still has a potentially slow convergence rate, and therefore, many iterations of the algorithm are required to arrive at a precise answer. The benefit of time consuming computations is the additional knowledge as to the distribution of the answer, but it comes at the cost of a significant growth in computation time. At 5 seconds an iteration, 500 iterations takes 1.38h. This is a significant growth in the computational effort required. Suggestions in the literature of 5000 iterations (13.8h) seem an unreasonable cost unless the additional information has significant value.

Generating Figure 5 took one hour while Figure 6 and Figure 7 took 1.5 hours which represents a linear but significant growth in the amount of time required to get an answer back from the image computing algorithm.

If the additional information is judged worthwhile, some consideration to optimization and parallelism of the algorithm is due. On the surface there appear to be limited opportunities for parallelism since each iteration depends on the previous iteration. One potential solution would be to run an initial set of iterations to exceed the burn-in period followed by many independent processors computing iterations following the jump-off point at the end of the burn-in. The results of all runs could then be accumulated to derive the final distribution.

6.4 Bias and Variance

One aspect that was not examined from the computed data sets, due to time constraints, was the bias and variance of the resulting MCMC runs when compared to the true images. It would be interesting to see the bias and variance of the solutions when comparing various regularization techniques as a means of judging the costs and benefits of those techniques. As well, for each of those techniques, it would be instructive to be able to see how the hyper-parameter selection affects the bias and variance in different regions of the image.

It would be expected, for example, that the bias of first-derivative discrete smoothing norms would be significant in regions of the image where there are rapid changes. One would expect that the results would be significantly different for regularization schemes with different characteristics such as Total Variation.

7 Conclusion

An algorithm for Markov Chain Monte Carlo employing a Gibbs Sampler was implemented based on a Tikhonov regularization scheme with a discrete smoothing norm that approximates a first-derivative. Images for three scenarios were generated: full angle projections with many projections, limited angle projections with limited projections, and full angle projections with limited projections. The images produced by the algorithm were of reasonable quality. The variance of the image was found to increase in some regions of the image where artifacts existed.

Computational requirements were found to be a limiting factor in the size of the images and length of the MCMC runs. The additional information in terms of hyper-parameter and image distributions provided interesting insights into the functioning of the MCMC algorithm, sources of error in the model, and the image and hyper-parameter distributions themselves.

References

[1] Kak A Slaney M, *Principles of Computerized Tomographic Imaging*, Society for Industrial and Applied Math, 2001

[2] Hansen P, *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*, SIAM monographs on mathematical modelling and computation, 1998

[3] Kaipio J Somersalo E, *Statistical and Computational Inverse Problems*, Springer-Verlag New York Inc, 2005

[4] Duda R Hart P Stork D, *Pattern Classification, 2nd Ed*, John Wiley & Sons Inc, 2001

[5] Computed Tomography, Wikipedia, http://en.wikipedia.org/wiki/Computed_tomography, visited Apr 8, 2009

[6] Radon Transform, Wikipedia, http://en.wikipedia.org/wiki/Radon_transform, visited Apr 8, 2009

[7] Monte Carlo method, Wikipedia, http://en.wikipedia.org/wiki/Monte_Carlo_method, visited Apr 10, 2009

[8] Gibbs Sampling, Wikipedia, http://en.wikipedia.org/wiki/Gibbs_sampling, visited Apr 12, 2009

[9] MCMC Tutorial, http://www.civs.ucla.edu/MCMC/MCMC_tutorial.htm, visited Apr 5, 2009

A Appendix: Code

This appendix contains the Octave code used to generate the images in this report. The code has not been run in MatLab but should be usable with some minor adjustments to syntax.

A.1 procedural

This code is the procedural sequence used to generate the figures in this report. It calls the `makeproj()` function. (See Appendix A.2.)

```
#! /usr/bin/octave --persist
% initial source
% from http://www.sce.carleton.ca/faculty/adler/elg7173/notes/elg7173-backprojection.html
% and heavily modified beyond all recognition

puts("Markov Chain Monte Carlo inverse solver for CT back projection\n");
puts(" (C) Alistair Boyle, 2009\n");

delta = 0.1; % noise variance = delta * mean projection
% hyper parameter for regularized image (-1 & LCURVE=auto)
hp_sel = 2; % = 1e-4 is the L-curve number
L_sel = 2; % regularization matrix 0=I 1=x-smooth 2,3=x-y-smooth 4=x-y-xy-smooth

% noise_proj_std is calculated (known)
noise_prior_std = delta/10;
angle_var=1; % amount of variance in angle estimates for model
K = 500;% how many iterations of the Gibbs sampler?

e=80; % number of elements in the image [ e x e ]
na=60; % number of projection angles to use

printf(' %dx%d image -> %dx%d reconstruction\n',e,e,e/2,e/2);
printf(' %d projection angles\n',na);

% Generate a sample image
function [H, x, y, img] = sensitivity_matrix(e,angles,rlim=1)
  r = linspace(-rlim,rlim,e);
  [x,y]= meshgrid(r,r);
  img = (x.^2 + y.^2) > rlim;
  img( x >.45 & x<.65 & y>-.05 & y<.45) =1;
  img( x >-.55 & x<-.25 & y>.45 & y<.65) =1;

  % calculate projections proj sample projections

  % create sensitivity matrix by building it from the projection matrices
  H = [];
  for ang= angles; %[0:ma/na:ma-0.1];
    prm= makeproj(ang*(pi/180),x,y);
    H = [H; prm];
  end
end

rlim=1;
ma=90; %180*(na-1)/na; % maximum angle to slice from (degrees)
angles=linspace(0,ma,na);
```

```
[H, x, y, img] = sensitivity_matrix(e,angles);
plen= size(x,1);

% create the projections from the sensitivity matrix
proj = H* img(:);

% add noise
noise=randn(size(proj));
noise_proj_std = delta*diag(mean(proj));
proj += noise_proj_std*noise;


% prevent an inverse crime: shrink to a courser grid
% reduce projections
r=2; % reduce by
convert=zeros(size(proj,1)/r,size(proj,1));
for i = 1:size(proj,1)/r
  convert(i,(i−1)*r+1:(i)*r) = ones(1,r)/r;
end
proj = convert*proj;

% generate new H matrix (model)
e=e/2;
[H, x, y] = sensitivity_matrix(e,angles);
plen= size(x,1);



function imbp = solve_inv(A, b, x, y, rlim, hp = 0, L = 0, x0 = 0)
  % backprojection reconstruction imbp (simple CT backprojection)
  % Unfiltered backprojection may be formulated as using the transpose of
  % the sensitivity matrix as the inverse.
  % Sensitivity Matrix (A) == Jacobian
  plen = prod(size(x));

  if(L == 0)
    L = eye(size(A));
  end
  if(x0 == 0)
    x0 = zeros(plen,1);
  end

  imbp = (hp^2*L'*L + A'*A)\(A'*b + hp^2*L'*L*x0);
end


imbp = solve_inv(H,proj,x,y,rlim);

L = speye(size(H,2)); % penalize total size of solution (L0)

% use 1d first deriv. approx
if(0) % for some reason octave doesnt like if,elif when the first if gets hit
elif(L_sel == 1) % x-smoothing
  L(1,1) = 0;
  for i = 2:plen*plen
    L(i,i−1) = −1;
  end
elif(L_sel == 2) % x-y-smoothing
  L = L*2;
  L(1:plen,1:plen) = 0;
  for i = plen+2:plen*plen
    L(i,i−1) = −1;
    L(i,i−plen) = −1;
  end
elif(L_sel == 3) % x-y-smoothing 2
  L = L*4;
  L(1:plen,1:plen) = 0;
  for i = plen+2:plen*plen−plen−1
    L(i,i−1) = −1;
    L(i,i+1) = −1;
    L(i,i−plen) = −1;
    L(i,i+plen) = −1;
  end
elif(L_sel == 4) % x-y-xy-smoothing
  L = L*8;
  L(1:plen,1:plen) = 0;
  for i = plen+2:plen*plen−plen−1
    L(i,i−1) = −1;
    L(i,i+1) = −1;
    L(i,i−plen) = −1;
    L(i,i+plen) = −1;
    L(i,i−1+plen) = −1;
    L(i,i+1+plen) = −1;
    L(i,i−1−plen) = −1;
    L(i,i+1−plen) = −1;
  end
end


% draw new hp
% x = current image
% hp0 = hyper-parameter prior
% n = number of elements in x
function hp_sel=draw_hp(x,hp0,n)
  hp=logspace(−10,2,500); % choose a range for hp
  p=hp.^(n+2).*exp(−1/2*hp*sum(x.^2)−1/2*(hp/hp0).^2+(n+2)/2*log10(hp));
  % get the CDF
  p=cumsum(p);
  p = p/p(end); % normalize
  % draw from a uniform distribution
  n=rand();
  s=find(p>n);
  hp_sel = hp(s(1));
end

prior = zeros(plen*plen,1); % not using a prior right now...

imbp_samples = zeros(plen*plen,K);
hp = ones(1,K)*hp_sel;
printf('Starting %d MCMC iterations...\n',K);
printf(' angle variance = %g\n',angle_var);
printf(' projection noise variance = %g\n',noise_proj_std^2);
printf(' prior variance = %g\n',noise_prior_std^2);
printf(' hyper parameter prior = %g\n',hp_sel);
puts(' ');
for k = 1:K
  printf("%d “,k);
  t=0;
  if(k==2)
    tic();
  end
  noise_proj = noise_proj_std*randn(size(proj)); % assuming we know the noise variance!
  if(k==1)
    noise_prior = 0;
  else
    noise_prior = noise_prior_std*randn(plen*plen,1); % assuming we know the noise variance!
    hp(k) = draw_hp(L/noise_prior_std*(imbp_samples(:,k-1)-prior),hp(1),plen*plen);
  end
  % add noise to angles in the model
  H = sensitivity_matrix(e, angles+angle_var*randn(size(angles)) );

  % calculate image from noisy projections and prior
  imbp_samples(:,k) = solve_inv(H/noise_proj_std,proj/noise_proj_std+noise_proj,x,y,rlim,hp(k),L/noise_prior_std,prior/noise_p
  if(k==2)
    printf('<-%gsx%d=%ds ',t=toc(),K,t*K)
  end
end
printf('\nMCMC completed.\n');
printf(' hyper parameter mode = %g\n',mode(hp));

function plot_img(img,T="",mask=[],range=[-0.1 1.1])
  plen=round(sqrt(size(img(:))(1))); % shouldn't need rounding...
  img= reshape( img, plen, plen); % turn it back into a square matrix
  if(size(mask) == size(img))
    img = img.*mask+(ones(size(mask))-mask);
  else
    printf('warning: plot_img %0s mask does not match image size\n',T);
  end
  if(size(range) != [1 2])
    range = [min(min(img)) max(max(img))];
  end
  hold on;
  imagesc([-1 1], [-1 1], img, range);
  xlabel(sprintf('min %g, max %g',range(1), range(2)));
  title(T);
end


imbp_reg = mean(imbp_samples')'; % equiv to MAP
imbp_var = (std(imbp_samples')').^2 ./ abs(imbp_reg); % standard deviation of the solution
% convert to log, so we can actually see something useful

figure();
colormap(bone);

% mask off this from the image
mask = ones(size(x));
mask((x.^2 + y.^2) >= rlim) = 0;

subplot(2,2,2);
```

```
plot_img(imbp_reg,'MAP', mask);

subplot(2,2,4);
plot_img(log10(imbp_var),sprintf('log10 normalized var, %d iterations',K),mask,[]);

subplot(2,2,1);
plot_img(img,'original',mask);

subplot(2,2,3);
hist(log10(hp),K/5);
title('log10(hyper parameter)');
xlabel(sprintf('mode %g',mode(hp)));

print('results.pdf');
      print('results.jpg');
```

A.2 makeproj

`makeproj()` creates a projection matrix for a particular angle. It is used in creating the sensitivity matrix for the forward model.

```
% Backprojection
% function proj = makeproj( a, x, y)
%
% The following function makeproj calculates an interpolating linear
% projection matrix for an angle a
%
% from http://www.sce.carleton.ca/faculty/adler/elg7173/notes/elg7173-backprojection.html

function proj = makeproj( a, x, y)
   if ~all(diff([size(x),size(y)])==0);
      error('x and y must be square');
   end

   rmax= max([abs(x(:));abs(y(:))]);
   spc = max(abs([mean(mean(diff(x'))), mean(mean(diff(y ))) ]));
   plen= size(x,1);

   %create x indices of projection ray
   xx=x*sin(a) + y*cos(a);
   xidx= (xx + rmax) / spc + 1;
   xi_l= floor(xidx);    xi_h = xi_l+1;
   xi_il= (xi_h-xidx);   xi_ih= (xidx-xi_l);

   %create y indices of projection ray
   yy=x*cos(a) - y*sin(a);
   yidx= (yy + rmax) / spc + 1;
   yi_l= floor(yidx);    yi_h = yi_l+1;
   yi_il= (yi_h-yidx);   yi_ih= (yidx-yi_l);

   % keep elements within bounds
   kp = (xx.^2 + yy.^2) < (rmax - spc);
   pnum = ones(plen,1) * (1:plen); pnum = pnum(kp);

   % create sparse interpolation matrix
```

```
posn = [ yi_l(kp), yi_h(kp), yi_l(kp), yi_h(kp)] + ...
  plen*([ xi_l(kp), xi_l(kp), xi_h(kp), xi_h(kp)]−1);
aprx = [xi_il(kp).*yi_il(kp), xi_il(kp).*yi_ih(kp), ...
         xi_ih(kp).*yi_il(kp), xi_ih(kp).*yi_ih(kp)];
proj = sparse( [pnum,pnum,pnum,pnum], posn, aprx,plen,plen^2);
```

A.3 circular pixels

Example code to find the area of a circle divided by a ray *s* `circle()`

```
#! /usr/bin/octave −q

function a = f(s,r=1)
  % s = the distance of the chord from the centre of the circle
  % r = the radius of the circle

  % theta = angle between the two lines that touch the edge of the circle
  % where the chord line touches the circle
  theta = 2.*acos(s./r);

  % area of the portion of the circle from 0 to theta radians
  ac = (theta/2)*(r.^2);

  % area of the two triangles that are formed from the centre of the circle
  % to the chrod line, where a right angle is formed by the intersection
  % of the chord and s
  at = 2*s.*(sqrt(r.^2−s.^2))/2;

  % area beyond the chord
  a = (ac − at);
end

f(linspace(−1,1,10))
s=linspace(−1,1,100);

plot(s,f(s)/pi);
xlabel('s/r');
ylabel('A');
      print('circle.pdf');
```