EEE 554 Random Signal Theory (Fall 2020)

Final Project: Image Manipulation Via MCMC

Due: December 3, 2020

In this project, you will use Markov Chain Monte Carlo (MCMC) to create abstract image mosaics from real photographs. Figure 1 shows an example: on the left is the original photo (of the Opera House in Sydney, Australia), and on the right is a mosaic of the same image constructed with MCMC.

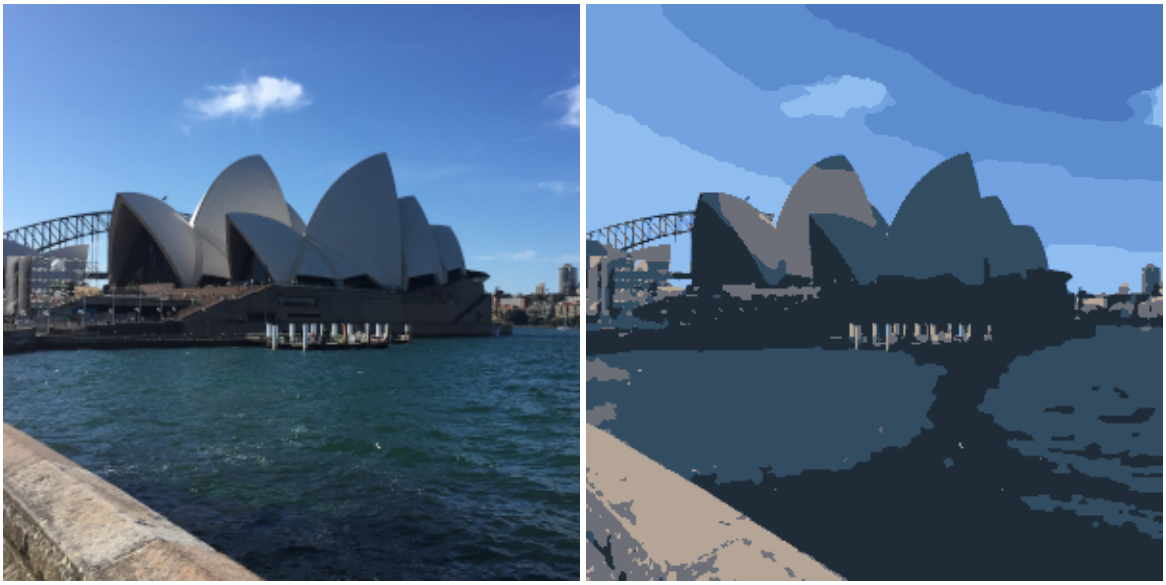

Figure 1: A real photograph on the left, and an abstract mosaic created from it using MCMC on the right.

First, read and understand this document. Then you will write code to implement MCMC, first to analyze a simple distribution, and then to do image manipulation. (As usual, Matlab is recommended but not required.) You should turn in a report which includes your well-documented code, as well as a write-up describing your implementation details, and the answers to specific questions asked below. Specific tasks that *must* be covered in your project report will be labeled by the bold word **Task**. To reiterate this: communication skills, combined with technical knowledge, are critical to any career in engineering. Part of the purpose of the projects in this class is for you to practice these communication skills by writing a report that clearly describes what you did and how you made your design decisions.

Your grade for the final project will be 50% based on the correctness of your implementations, and 50% based on the clarity of your project report. You may discuss the project in broad terms with other students, but you must work independently.

1 MCMC and the Metropolis-Hastings Algorithm

The basic theory of MCMC was covered in Recitation 12 on November 12. You are encouraged to watch this recitation video if you haven't already. MCMC is a technique to generate samples from a target distribution $p_X(x)$ or $f_X(x)$ (the distribution can be discrete or continuous). The idea is to set up a homogenous Markov chain $X[n]$ with stationary distribution equal to the target distribution. Thus, the Markov chain can be initialized at any value, then propagated forward long enough for the Markov chain to mix, at which point it will generate samples from the target distribution.

There are several algorithms that implement the MCMC idea. The specific algorithm we will use in this project is called Metropolis-Hastings, which is outlined in the following pseudocode.

Input: Initial value $X[0]$, target distribution $p_X(x)$, and a conditional distribution $g(x'|x)$ where $\sum_{x'} g(x'|x) = 1$ for all x, and where $g(x'|x) = g(x|x')$.
Output: Markov chain $X[n]$ with stationary distribution $p_X(x)$
for $n = 1$ *to* N **do**
  $x \leftarrow X[n-1]$;
  $X' \sim g(x'|x)$;
  $\alpha \leftarrow \frac{p_X(X')}{p_X(x)}$;
  $U \sim \mathcal{U}(0,1)$;
  if $U < \alpha$ **then**
    $X[n] \leftarrow X'$;
  else
    $X[n] \leftarrow X[n-1]$;
  end
end

A few notes about this algorithm:

1. The pseudocode above is written for a discrete distribution p_X, but the algorithm can also be used for a continuous distribution f_X. The only changes required are to replace the ratio of PMFs $\frac{p_X(x')}{p_X(x)}$ with the ratio of PDFs $\frac{f_X(x')}{f_X(x)}$, and to replace the conditional PMF $g(x'|x)$ with a conditional PDF; i.e., $\int g(x'|x)dx' = 1$ for all x. (It is still required that $g(x'|x) = g(x|x')$.)

2. Metropolis-Hastings only requires the calculation of the ratio $\frac{p_X(x')}{p_X(x)}$. Thus, if it is easier to compute a function $h(x)$ where $p_X(x) = c\,h(x)$ for some constant c, then it is equally good to use the ratio $\frac{h(x')}{h(x)}$.

3. While it is technically true that Metropolis-Hastings yields a Markov chain $X[n]$ with the correct stationary distribution no matter what $g(x'|x)$ is used, in practice the choice of this conditional distribution has a big impact on the efficiency of the algorithm. First, a conditional distribution $g(x'|x)$ should be chosen that is easy to generate samples from. Second, $g(x'|x)$ can impact how quickly the Markov chain mixes; if it mixes too slowly, then the algorithm must be run for a long time for the samples produced in $X[n]$ to be from the correct distribution. One of the disadvantages of MCMC is that it is usually hard to know for sure whether the Markov chain has mixed. Even so, MCMC is very useful in numerous applications.

2 Implement Metropolis-Hastings on a Continuous Scalar Distribution

To become familiar with Metropolis-Hastings, you will first implement it for a continuous distribution of a single random variable.

Task 1 *Implement Metropolis-Hastings for the same continuous distribution that was assigned to you in the midterm project. To initialize the Markov chain, set $X[0] = x_0$ for a constant x_0 of your choice. Use $g(x'|x) = \mathcal{N}(x, 1)$. Explain why this conditional distribution satisfies $g(x'|x) = g(x|x')$. Run the algorithm to generate at least 10,000 samples. Use these samples to plot an estimated PDF, and compare against the true PDF.*

Next, to understand how quickly this Markov chain mixes, you will conduct some experiments to understand the PDF of $X[n]$ for specific values of n.

Task 2 *With the same initialization and choice of g as in the previous task, estimate and plot the PDF of $X[n]$ for $n = 1, 3, 10, 30, 100$. Note that this requires generating multiple samples of $X[n]$ for each of these n values. For example, to generate multiple samples of $X[1]$, initialize the Markov chain to $X[0] = x_0$, run the algorithm for one step to generate $X[1]$, then initialize the Markov chain again to generate the next sample, and so on. When you re-initialize the Markov chain, be sure **not to** reset the seed for the random number generator, or else you will get the sample of $X[1]$ over and over again. Does it seem as if the PDF of $X[n]$ approaches f_X as $n \to \infty$?*

How quickly the Markov chain mixes depends on the choice of g. Next, you will try different conditional distributions g to see if this improves the mixing time.

Task 3 *Repeat Task 2 for each of the following conditional distributions g:*

1. $g(x'|x) = \mathcal{N}(x, 10)$
2. $g(x'|x) = \mathcal{N}(x, 0.1)$
3. $g(x'|x) = \mathcal{U}(x - 0.5, x + 0.5)$

Also, confirm that each of these distributions satisfy $g(x'|x) = g(x|x')$. Which choice seems to mix fastest?

3 Choose and Load an Image

You will create an abstract mosaic by starting from a real photograph. If you can find one that you like from your personal collection, great! Otherwise, feel free to choose a photograph from any source. It is best to choose an image that has multiple shapes of somewhat flat color, so that when it is converted into a mosaic that has only a few colors, the basic shape of the image will be preserved. Crop and/or scale the image so that its resolution is 300×300 pixels. (You may experiment with smaller or larger resolutions, but it is recommended to start at this size.)

Once you have chosen an image, you need to load it into your computing environment. This can take some time to set up correctly. In Matlab, use the function `imread` to load an image file. By default, `imread` outputs a three-dimensional array of dimension $m \times n \times 3$, where $m \times n$ is the resolution of the image, and the third dimension holds the red, green, and blue channels. Images can be displayed in Matlab using the command `image`, which accepts input

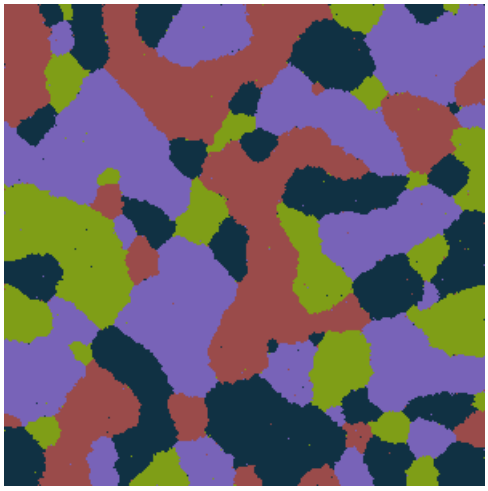

Figure 2: An example of a mosaic image created without using a real photograph as a reference.

arrays in the same format that `imread` produces. Images can be saved to a file using the command `imwrite`. One important note is that, by default, `imread` produces an array with values stored in the `uint8` format (unsigned 8-bit integers). To convert an array `A` to standard floating point for easier manipulation, use the command `double(A)`; to convert back use `uint8(A)`.

Task 4 *Choose an image. Write code to load it into your computing environment and then display it.*

4 The Distribution for a Mosaic Image

In order to produce mosaics images as in Figure 1, we first need to design a distribution that models the characteristics of these images. At first, ignore the requirement that the mosaic image approximates a real photograph, then later you will add this functionality. Figure 2 shows an example of a mosaic image that is not based on any existing photograph. The mosaic distribution will not be a simple scalar distribution like the one that you analyzed above, but rather a complicated distribution composed of multiple random variables. The Metropolis-Hastings algorithm can be generalized to work with any random vector $\mathbf{X}$, rather than a scalar random variable X.

A mosaic image consists of contiguous regions of colors selected from a small number of color options (3–15 depending on preference). "Contiguous" means that two neighboring pixels usually have exactly the same color. We can model this probabilistically in the following way. The image will be represented by two parts: a color palette, and a color index for each pixel. The color palette consists of k colors, each of which consists of 3 numbers representing the red, green, and blue channels. That is, the palette consists of $3k$ random variables denoted P_{ij} where $i = 1, \ldots, k$ and $j = 1, 2, 3$. Each of the palette numbers should be between 0 and 255. The color index for each pixel is given by Z_{ij} for the pixel at position (i, j), where $i = 1, \ldots, m$ and $j = 1, \ldots, n$, assuming $m \times n$ is the resolution of the image. Each of the index numbers $Z_{ij} \in \{1, 2, \ldots, k\}$ indicates which color from the color palette this pixel takes on. That is, $Z_{ij} = \ell$ means that the color of the pixel at position (i, j) has RGB representation $(P_{\ell,1}, P_{\ell,2}, P_{\ell,3})$. Denote $\mathbf{P}$ to be the vector of all the palette variables, and $\mathbf{Z}$ the vector of all index variables. Putting this together, the random vector

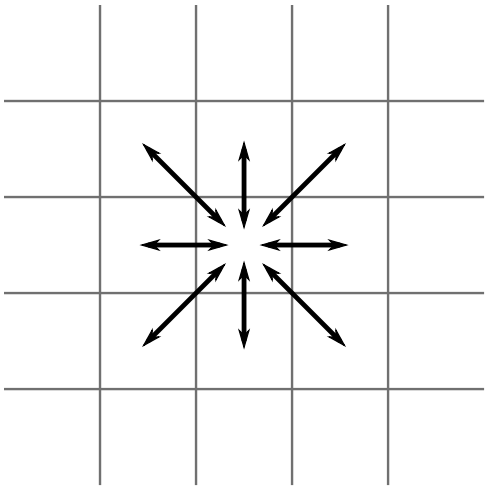

Figure 3: A representation of neighboring pixels in an image. Each square is a pixel, and the neighbors of the central pixel are those indicated by the arrows.

$\mathbf{X} = [\mathbf{P}, \mathbf{Z}]$ represents the entire mosaic image. Note that, in order to display or save the image, you will need to convert $\mathbf{X}$ into an $m \times n \times 3$ image array containing the color of each pixel.

We still need to specify the distribution of these variables. Assume the color palette variables P_{ij} are uniform on the interval $(0, 255)$. The index variables Z_{ij} should have a distribution such that neighboring pixels are more likely to have the same index than not. We can set this up as follows:

$$p_{\mathbf{Z}}(\mathbf{z}) = c \, \exp\left\{ r \sum_{i,j,i',j' : (i,j) \text{ is a neighbor of } (i',j')} \delta[z_{ij} - z_{i'j'}] \right\}.$$

Recall that $\delta[0] = 1$ and $\delta[k] = 0$ for $k \neq 0$; thus, the summation inside the exponential counts the number of neighboring pixels (i, j) and (i', j') that have the same color index. That is, this distribution says that it is *more likely* that neighboring pixels have the same color than not. The constant r is a parameter that you must choose: larger r means whether neighboring pixels are the same has a stronger effect on the probability. The constant c is only there to ensure that this is a proper distribution; to implement Metropolis-Hastings, you do not need to know c.

We still haven't specified exactly what "neighboring" pixels are. For these purposes, we define neighboring pixels as any two pixels that are adjacent horizontally, vertically, *or* diagonally. See the diagram in Figure 3.

You will implement Metropolis-Hastings to create samples from this distribution on mosaic images. To do this, you need to carefully select the conditional distribution $g(\mathbf{x}'|\mathbf{x})$. Recall that the vector of random variables $\mathbf{X}$ consists of the P_{ij} variables and the Z_{ij} variables. The conditional distribution $g(\mathbf{x}'|\mathbf{x})$ amounts to a way to choose a new guess $\mathbf{x}'$ from an existing guess $\mathbf{x}$. This needs to be done in a way so that the distribution is symmetric (i.e., $g(\mathbf{x}'|\mathbf{x}) = g(\mathbf{x}|\mathbf{x}')$), so that it is easy to sample from, and so that the entire space will be eventually explored. A good way to implement a conditional distribution $g(\mathbf{x}'|\mathbf{x})$ is as follows. First, randomly choose one among all of these variables, and then randomly adjust only that single variable. Note that Z_{ij} are discrete variables, since they only take values in $\{1, \ldots, k\}$, while P_{ij} are continuous, since they can take any value in the interval $[0, 255]$. You will need to find a good way to adjust either one of the Z_{ij} variables or one of the P_{ij} variables. Be sure to do so in a way that the conditional distribution

is symmetric. Note that, even though you will only adjust one of these variables at a time, since the identity of the variable is chosen randomly, you will still fully explore the space of all possibly images.

Task 5 *Implement Metropolis-Hastings to produce a sample from the mosaic image distribution for a* 300×300 *image with* $k = 4$ *colors. Initialize your Markov chain by randomly selecting* $P_{ij} \sim \mathcal{U}(0, 255)$ *and* Z_{ij} *uniformly from* $\{1, 2, \ldots, k\}$*. You will need to experiment to find a good value for the constant* r*. Note that for this task you do* ***not*** *need to store the entire chain. Rather, you only want to produce a* ***single sample*** *from the distribution. Thus, you only need to run the algorithm long enough for the Markov chain to mix, and keep the last value; this will probably take millions of iterations. You will never truly know if your Markov chain has mixed, but if you can produce an image roughly like Figure 2, then your algorithm is working correctly. You may find it helpful to regularly display the current image to see if it looks right. Include the resulting mosaic image in your report.*

When implementing your algorithm, you will need to be careful to make your code as efficient as possible, or else the algorithm will take too long to mix. In particular, you should consider how to optimally calculate the ratio $\frac{p_{\mathbf{X}}(x')}{p_{\mathbf{X}}(x)}$*. It may be helpful to* ***profile*** *your code. In Matlab, this can be done by clicking the "Run and Time" button. This will run the code as usual, and then produce an interactive dataset that tells you how much time the code spent in which sections. If your code is taking surprisingly long at certain operations, you should think about how to make those operations more efficient. (If you are not using Matlab, most other computing environments also have a mechanism to profile your code.)*

5 Incorporating Your Image

We now need to adjust the probability distribution so that the resulting mosaic image looks roughly like your chosen image. Do this by adding a term to the probability distribution which encourages the color of each pixel in the mosaic image to be close to that in the original image. Let $A_{ij\ell}$ be the value of your image at pixel (i, j) in channel ℓ, where $\ell = 1, 2, 3$. Also let $C_{ij\ell}(\mathbf{p}, \mathbf{z})$ be the corresponding value for the mosaic image with color palette variables $\mathbf{p}$ and index variables $\mathbf{z}$. The distribution is given by

$$p_{\mathbf{P},\mathbf{Z}}(\mathbf{p}, \mathbf{z}) = c\, f_{\mathbf{P}}(\mathbf{p})\, p_{\mathbf{Z}}(\mathbf{z})\, \exp\left\{ -s \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{\ell=1}^{3} \left| C_{ij\ell}(\mathbf{p}, \mathbf{z}) - A_{ij\ell}) \right| \right\}.$$

Here, $f_{\mathbf{P}}$ and $p_{\mathbf{Z}}$ are the same distributions on $\mathbf{P}$ and $\mathbf{Z}$ as above; s is a parameter that you must choose; and again c is a constant to make this a proper distribution, which you do not need to worry about. Note that this probability will be larger if $C_{ij\ell}(\mathbf{p}, \mathbf{z})$ is closer to $A_{ij\ell}$.

Task 6 *Adjust your Metropolis-Hastings algorithm to incorporate the real image, and produce a mosaic image like that in Figure 1 that preserves the basic shape of the image with only a small number of colors. The number of colors* k *is your choice: choose a number that looks good to you! Be sure to properly document all the decisions you make in your implementation.*

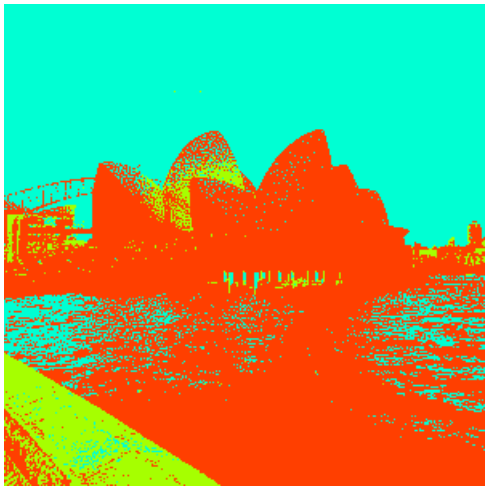

Figure 4: A different abstract image made starting from the same original image as in Figure 1.

6 Make Something Different

MCMC is a powerful tool for this kind of data manipulation. By choosing different distributions on the color palette and the color indexes, you can create all sorts of different images. An example is shown in Figure 4, made from the same original image as in Figure 1; this one was made using MCMC with a slightly different underlying distribution, but I won't tell you exactly how. For the final task, you will explore these possibilities.

Task 7 *Make a different kind of abstract image via MCMC. Be sure to clearly describe what you did. Be creative! Bonus points will be given for especially cool results. Some ideas to consider (but do not feel limited by these):*

1. *Change the distribution of the color palette.*
2. *Change the relationship between color indexes in neighboring pixels.*
3. *Change the definition of "neighboring" pixels.*
4. *Change what it means for the mosaic color and the real photo color to be "close".*