Image Restoration: Richardson Lucy Algorithm

Arijit Dutta Aurindam Dhar Kaustav Nandy

November 1, 2010

Basic Idea

Restoration of Images

- Restoration of digital images from their degraded measurement has always been a problem of great interest.
- A specific solution to the problem of image restoration is generally determined by the nature of degradation phenomena.
- So, it is highly dependent of the nature of the noise present there.
- Obviously, one has to determine the nature of the noise first.

Basic Idea

Restoration of Images

- Restoration of digital images from their degraded measurement has always been a problem of great interest.
- A specific solution to the problem of image restoration is generally determined by the nature of degradation phenomena.
- So, it is highly dependent of the nature of the noise present there.
- Obviously, one has to determine the nature of the noise first.

Basic Idea

Restoration of Images

- Restoration of digital images from their degraded measurement has always been a problem of great interest.
- A specific solution to the problem of image restoration is generally determined by the nature of degradation phenomena.
- So, it is highly dependent of the nature of the noise present there.
- Obviously, one has to determine the nature of the noise first.

Basic Idea

Restoration of Images

- Restoration of digital images from their degraded measurement has always been a problem of great interest.
- A specific solution to the problem of image restoration is generally determined by the nature of degradation phenomena.
- So, it is highly dependent of the nature of the noise present there.
- Obviously, one has to determine the nature of the noise first.

Basic Idea

Restoration of Images

- Restoration of digital images from their degraded measurement has always been a problem of great interest.
- A specific solution to the problem of image restoration is generally determined by the nature of degradation phenomena.
- So, it is highly dependent of the nature of the noise present there.
- Obviously, one has to determine the nature of the noise first.

Continued...

What is an Image?

- An image is nothing but a huge matrix of numbers.
- Those numbers are just the pixel values of the corresponding points in the image.

Point Spread Function

- The *Point Spread Function* describes the response of an imaging system to a point source or point object.

Continued...

What is an Image?

- An image is nothing but a huge matrix of numbers.
- Those numbers are just the pixel values of the corresponding points in the image.

Point Spread Function

- The *Point Spread Function* describes the response of an imaging system to a point source or point object.

Continued...

What is an Image?

- An image is nothing but a huge matrix of numbers.
- Those numbers are just the pixel values of the corresponding points in the image.

Point Spread Function

- The *Point Spread Function* describes the response of an imaging system to a point source or point object.

CONTINUED...

What is an Image?

- An image is nothing but a huge matrix of numbers.
- Those numbers are just the pixel values of the corresponding points in the image.

Point Spread Function

- The *Point Spread Function* describes the response of an imaging system to a point source or point object.

Point Spread Function(PSF)

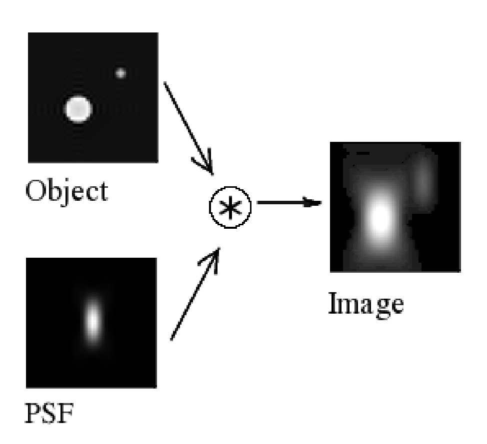

Figure: Example of PSF

Richardson-Lucy Algorithm

- Given the point spread function, Richardson-Lucy Algorithm provides an iterative method of image restoration.
- This algorithm was introduced by W.H. Richardson (1972) and L.B. Lucy (1974).
- This is also known as Richardson-Lucy Deconvolution.
- This is a Bayesian Based Iterative Method of image restoration.

Richardson-Lucy Algorithm

- Given the point spread function, Richardson-Lucy Algorithm provides an iterative method of image restoration.
- This algorithm was introduced by W.H. Richardson (1972) and L.B. Lucy (1974).
- This is also known as Richardson-Lucy Deconvolution.
- This is a Bayesian Based Iterative Method of image restoration.

Richardson-Lucy Algorithm

- Given the point spread function, Richardson-Lucy Algorithm provides an iterative method of image restoration.
- This algorithm was introduced by W.H. Richardson (1972) and L.B. Lucy (1974).
- This is also known as Richardson-Lucy Deconvolution.
- This is a Bayesian Based Iterative Method of image restoration.

Richardson-Lucy Algorithm

- Given the point spread function, Richardson-Lucy Algorithm provides an iterative method of image restoration.
- This algorithm was introduced by W.H. Richardson (1972) and L.B. Lucy (1974).
- This is also known as Richardson-Lucy Deconvolution.
- This is a Bayesian Based Iterative Method of image restoration.

Brief Description

Suppose,

Y : Degraded Image,

Λ : Original Image,

P : Point Spread Function,

$*$: Operation of Convolution.

Then,

$$Y = \Lambda * P$$

Comment

Numerical values of Y, Λ and P can be considered as a measure of frequency at that point.

Brief Description

Suppose,

Y : Degraded Image,

Λ : Original Image,

P : Point Spread Function,

$*$: Operation of Convolution.

Then,

$$Y = \Lambda * P$$

Comment

Numerical values of Y, Λ and P can be considered as a measure of frequency at that point.

Continued...

In the model,

$$Y = \Lambda * P$$

- $P = \left(\left(p(i,j)\right)\right)$, $p(i,j) =$ P[Photon emitted at i is seen at j]
- $\Lambda = (\lambda_1, \ldots, \lambda_n)'$, $\lambda_i =$ True pixel value at the i^{th} point.
- $Y = (y_1, \ldots, y_d)'$, $y_j =$ Observed pixel value at the j^{th} point.

Comment

For simplicity, we may assume that $d = n$, i.e. the observed and the true images contain the same number of pixels.

Continued...

In the model,

$$Y = \Lambda * P$$

- $P = \left(\left(\, p(i,j) \,\right)\right)$, $p(i,j) =$ P[Photon emitted at i is seen at j]
- $\Lambda = (\lambda_1, \ldots, \lambda_n)'$, $\lambda_i =$ True pixel value at the i^{th} point.
- $Y = (y_1, \ldots, y_d)'$, $y_j =$ Observed pixel value at the j^{th} point.

Comment

For simplicity, we may assume that $d = n$, i.e. the observed and the true images contain the same number of pixels.

Continued...

In the model,

$$Y = \Lambda * P$$

- $P = \Big(\Big(p(i,j)\Big)\Big)$, $p(i,j) =$ P[Photon emitted at i is seen at j]
- $\Lambda = (\lambda_1, \ldots, \lambda_n)'$, $\lambda_i =$ True pixel value at the i^{th} point.
- $Y = (y_1, \ldots, y_d)'$, $y_j =$ Observed pixel value at the j^{th} point.

Comment

For simplicity, we may assume that $d = n$, i.e. the observed and the true images contain the same number of pixels.

Continued...

In the model,

$$Y = \Lambda * P$$

- $P = \left(\left(p(i,j)\right)\right)$, $p(i,j) =$ P[Photon emitted at i is seen at j]
- $\Lambda = (\lambda_1, \ldots, \lambda_n)'$, $\lambda_i =$ True pixel value at the i^{th} point.
- $Y = (y_1, \ldots, y_d)'$, $y_j =$ Observed pixel value at the j^{th} point.

Comment

For simplicity, we may assume that $d = n$, i.e. the observed and the true images contain the same number of pixels.

Continued...

In the model,

$$Y = \Lambda * P$$

- $P = \Big(\Big(\, p(i,j) \,\Big)\Big)$, $p(i,j) =$ P[Photon emitted at i is seen at j]
- $\Lambda = (\lambda_1, \ldots, \lambda_n)'$, $\lambda_i =$ True pixel value at the i^{th} point.
- $Y = (y_1, \ldots, y_d)'$, $y_j =$ Observed pixel value at the j^{th} point.

✍ Comment

For simplicity, we may assume that $d = n$, i.e. the observed and the true images contain the same number of pixels.

Distributions: Observed Pixels

- Notice that y_j is nothing but the count of the photon seen at j.
- So y_j has a Poisson distribution.
- In fact,

$$y_j \sim \text{Poisson}(\mu_j)$$

where,

$$\mu_j = \sum_{i=1}^{n} \lambda_i \, p(i,j)$$

Distributions: Observed Pixels

- Notice that y_j is nothing but the count of the photon seen at j.
- So y_j has a Poisson distribution.
- In fact,

$$y_j \sim \text{Poisson}(\mu_j)$$

where,

$$\mu_j = \sum_{i=1}^{n} \lambda_i \, p(i,j)$$

Distributions: Observed Pixels

- Notice that y_j is nothing but the count of the photon seen at j.
- So y_j has a Poisson distribution.
- In fact,

$$y_j \sim \text{Poisson}(\mu_j)$$

where,

$$\mu_j = \sum_{i=1}^{n} \lambda_i \, p(i,j)$$

Distributions: Observed Pixels

- Notice that y_j is nothing but the count of the photon seen at j.
- So y_j has a Poisson distribution.
- In fact,

$$y_j \sim \text{Poisson}(\mu_j)$$

where,

$$\mu_j = \sum_{i=1}^{n} \lambda_i \, p(i, j)$$

Distribution: Spread Function

- The distribution of spread function may vary from problem to problem.
- In our problem, we have taken Gaussian spread function which is given by:

$$p(i,j) = exp\left(-\frac{d(i,j)^2}{\sigma^2}\right)$$

where, $d(i,j) =$ Distance between i and j

Remark

In case of multidimension, standard Euclidean norm is taken as a measure of distance.

Distribution: Spread Function

- ☞ The distribution of spread function may vary from problem to problem.
- ☞ In our problem, we have taken Gaussian spread function which is given by:

$$p(i,j) = exp\left(-\frac{d(i,j)^2}{\sigma^2}\right)$$

where, $d(i,j)$ = Distance between i and j

✍Remark

In case of multidimension, standard Euclidean norm is taken as a measure of distance.

Distribution: Spread Function

- The distribution of spread function may vary from problem to problem.
- In our problem, we have taken Gaussian spread function which is given by:

$$p(i,j) = exp\left(-\frac{d(i,j)^2}{\sigma^2} \right)$$

where, $d(i,j) =$ Distance between i and j

Remark

In case of multidimension, standard Euclidean norm is taken as a measure of distance.

Distribution: Spread Function

- ☞ The distribution of spread function may vary from problem to problem.
- ☞ In our problem, we have taken Gaussian spread function which is given by:

$$p(i,j) = exp\left(- \frac{d(i,j)^2}{\sigma^2} \right)$$

where, $d(i,j) =$ Distance between i and j

✍Remark

In case of multidimension, standard Euclidean norm is taken as a measure of distance.

Description of the Algorithm

- Define, the contribution of λ_i on y_j as

$$z(i,j) \sim \text{Poisson}\Big(\lambda_i \, p(i,j)\Big)$$

- Then,

$$y_j = \sum_{i=1}^{n} z(i,j)$$

and

$$\frac{\lambda_i \, p(i,j)}{\sum_k \lambda_k \, p(k,j)}$$

is the proportion of y_j emitted by i.

Description of the Algorithm

- Define, the contribution of λ_i on y_j as

$$z(i,j) \sim \text{Poisson}\Big(\lambda_i \, p(i,j)\Big)$$

- Then,

$$y_j = \sum_{i=1}^{n} z(i,j)$$

and

$$\frac{\lambda_i \, p(i,j)}{\sum_k \lambda_k \, p(k,j)}$$

is the proportion of y_j emitted by i.

Description of the Algorithm

- Define, the contribution of λ_i on y_j as

$$z(i,j) \sim \text{Poisson}\Big(\lambda_i \, p(i,j)\Big)$$

- Then,

$$y_j = \sum_{i=1}^{n} z(i,j)$$

and

$$\frac{\lambda_i \, p(i,j)}{\sum_k \lambda_k \, p(k,j)}$$

is the proportion of y_j emitted by i.

Continued...

- If we know Λ then $z(i,j)$ is estimated by:

$$\hat{z}(i,j) = \frac{y_j \, \lambda_i \, p(i,j)}{\sum_k \lambda_k \, p(k,j)}$$

- Given $\hat{z}(i,j)$, λ_i is estimated by:

$$\hat{\lambda}_i = \sum_{j=1}^{d} \hat{z}(i,j)$$

Continued...

- If we know Λ then $z(i,j)$ is estimated by:

$$\hat{z}(i,j) = \frac{y_j \, \lambda_i \, p(i,j)}{\sum_k \lambda_k \, p(k,j)}$$

- Given $\hat{z}(i,j)$, λ_i is estimated by:

$$\hat{\lambda}_i = \sum_{j=1}^{d} \hat{z}(i,j)$$

CONTINUED...

So, ultimately it gives an iterative procedure:

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} \sum_{j=1}^{d} \frac{y_j \, p(i,j)}{\sum_k \lambda_k^{(t)} p(k,j)} \qquad \ldots\ldots(\ast)$$

Back

E-M and Richardson Lucy Algorithms

- Here, $z(i,j)$'s are complete data and y_j's are random.
- So, $z(i,j) \mid y_j \sim \text{Bin}\Big(y_j, p_*(i,j)\Big)$, where $p_*(i,j) = \frac{\lambda_i\, p(i,j)}{\sum_k \lambda_k\, p(k,j)}$
- $p(i,j)$'s are given to us; our aim is to estimate λ_i's.
- By E-M algorithm, first we will calculate

$$\arg\max_{\lambda} E\left[\log f_{z(i,j)}(\underset{\sim}{\lambda}) \mid y_j, \underset{\sim}{\lambda}^0\right]$$

$$= \arg\max_{\lambda} E\Bigg[\Bigg\{\log\left(\frac{y_i}{z(i,j)}\right) + z(i,j)\log p_*^0(i,j)$$

$$+ (y_i - z(i,j))\log\left(1 - p_*^0(i,j)\right)\Bigg\} \Bigg| \; y_j, \underset{\sim}{\lambda}^0\Bigg]$$

where $\underset{\sim}{\lambda}^0$ is some initial estimate of $\underset{\sim}{\lambda}$.

Continued...

- Now, $E[z(i,j) \mid y_j, \underset{\sim}{\lambda}] = y_j \, p_*(i,j) = \hat{z}(i,j)$, say.
- Also,

$$\sum_{j=1}^{d} z(i,j) \sim \text{Poisson}(\lambda_i) \quad \text{since} \sum_{j=1}^{d} p(i,j) = 1$$

- So, we can have an estimate of λ_i as $\hat{\lambda}_i = \sum_{j=1}^{d} \hat{z}(i,j)$.
- Following similar steps, at the $(t+1)^{th}$ iteration, we will have,

$$\hat{\lambda}_i^{(t+1)} = \sum_{j=1}^{d} \hat{z}^{(t)}(i,j) = \hat{\lambda}_i^{(t)} \sum_{j=1}^{d} \frac{y_j \, p(i,j)}{\sum_k \hat{\lambda}_k^{(t)} p(k,j)}$$

- The above is exactly what we have obtained from Richardson Lucy algorithm.

Computer Implementation
Problems

- Suppose, we are given a square image of size $M \times M$.
- Then there is a total of M^2 pixels.
- For each of the pixels, we have to apply the algorithm.
- To compute the denominator of ✱, we have to run a loop over all M^2 pixels.
- This denominator is to be calculated for each of the M^2 terms in the outer most sum of ✱.
- So, for a single iteration step, complexity will be $M^2 \times M^2 \times M^2 = M^6$.
- Now, even a small image is of 256×256 or 512×512. So, first we have to reduce the complexity.

▸ R-L algorithm

Reducing Complexity

- First, notice that photon emitted from a particular point affects the nearby points most.
- In fact, as the distance between i and j increases, $p(i,j)$ tends to 0.
- So, for a fixed i, we should run the loop over only the range of j for which $p(i,j) > 0$.
- This will reduce the complexity significantly.
- We have implemented this algorithm in **C**.

Reducing Complexity

- First, notice that photon emitted from a particular point affects the nearby points most.
- In fact, as the distance between i and j increases, $p(i,j)$ tends to 0.
- So, for a fixed i, we should run the loop over only the range of j for which $p(i,j) > 0$.
- This will reduce the complexity significantly.
- We have implemented this algorithm in $\mathbf{C}$.

Reducing Complexity

- First, notice that photon emitted from a particular point affects the nearby points most.
- In fact, as the distance between i and j increases, $p(i, j)$ tends to 0.
- So, for a fixed i, we should run the loop over only the range of j for which $p(i, j) > 0$.
- This will reduce the complexity significantly.
- We have implemented this algorithm in C.

Reducing Complexity

- First, notice that photon emitted from a particular point affects the nearby points most.
- In fact, as the distance between i and j increases, $p(i,j)$ tends to 0.
- So, for a fixed i, we should run the loop over only the range of j for which $p(i,j) > 0$.
- This will reduce the complexity significantly.
- We have implemented this algorithm in $\mathbf{C}$.

Reducing Complexity

- First, notice that photon emitted from a particular point affects the nearby points most.
- In fact, as the distance between i and j increases, $p(i, j)$ tends to 0.
- So, for a fixed i, we should run the loop over only the range of j for which $p(i, j) > 0$.
- This will reduce the complexity significantly.
- We have implemented this algorithm in $\mathbf{C}$.

Output

Figure: Blurred Image

Figure: Restored Image

Comparison

Figure: Restored Image

Figure: Original Image

Output

Figure: Blurred Image

Figure: Restored Image

Comparison

Figure: Restored Image

Figure: Original Image

Problem That Remains

- The restoration is mediocre.
- To be more specific, it is very bad near the portion where there is high contrast.
- The implemented algorithm takes a lot of time to run.

Acknowledgement

- Dr. Deepayan Sarkar, ISI Delhi.
- Wikipedia

THANK YOU