An Introduction to Stochastic Methods in Optimization (Revised, Oct. 24th, 2012)

Marios S. Pattichis

UNM

Outline

Outline

Where to begin ...

A nice introduction to Simulation and Simulated Annealing is given by Sheldon Ross in *Simulation*. We will refer to this book by [SR].

A more advanced introduction with answers to many basic questions is given in Brian D. Ripley's *Stochastic Simulation*. We will refer to this book by [BR].

For image processing students, we need [BR] , who is also an authority in Spatial Statistics. He brings examples from image processing also.

Where to begin ...

A nice introduction to Simulation and Simulated Annealing is given by Sheldon Ross in *Simulation*. We will refer to this book by [SR].

A more advanced introduction with answers to many basic questions is given in Brian D. Ripley's *Stochastic Simulation*. We will refer to this book by [BR].

For image processing students, we need [BR] , who is also an authority in Spatial Statistics. He brings examples from image processing also.

Where to begin ...

A nice introduction to Simulation and Simulated Annealing is given by Sheldon Ross in *Simulation*. We will refer to this book by [SR].

A more advanced introduction with answers to many basic questions is given in Brian D. Ripley's *Stochastic Simulation*. We will refer to this book by [BR].

For image processing students, we need [BR] , who is also an authority in Spatial Statistics. He brings examples from image processing also.

Goals

We have the following two reasons from [BR]:

- Optimization from noisy observations.:
  The basic idea is that we can only approximate the function to be optimized. Here, let us note that we now also have more methods to deal with noisy measurements. Many researchers suggest using Stochastic Methods when the optimization function is not differentiable. This is similar to this category.
- Efficient Exploratory Optimization:
  The basic idea is to define a probability distribution over the space that we are optimizing and then draw random samples over that space and evaluate the function. For example, the standard practice of uniform sampling is very similar to drawing samples from a uniform distribution.

Stochastic Optimization is not recommended when we know a lot about the problem [BR].

Goals

We have the following two reasons from [BR]:

- Optimization from noisy observations.:
  The basic idea is that we can only approximate the function to be optimized. Here, let us note that we now also have more methods to deal with noisy measurements. Many researchers suggest using Stochastic Methods when the optimization function is not differentiable. This is similar to this category.
- Efficient Exploratory Optimization:
  The basic idea is to define a probability distribution over the space that we are optimizing and then draw random samples over that space and evaluate the function. For example, the standard practice of uniform sampling is very similar to drawing samples from a uniform distribution.

Stochastic Optimization is not recommended when we know a lot about the problem [BR].

Outline

Optimization Problem Definition Without Derivatives

Without requiring differentiability, the basic optimization problem is given by:

Find $x^* \in D$ for some domain $D \subset R^n$ such that:

$$f(x^*) = \sup_{x \in D} f(x).$$

A local maximum is found when $\hat{x}$ satisfies $f(x) < f(\hat{x})$ for all $x \in D$ for some neighborhood defined by $0 < \|x - \hat{x}\| \leq \delta$, some δ.

Optimization Problem Definition Without Derivatives

Without requiring differentiability, the basic optimization problem is given by:
Find $x^* \in D$ for some domain $D \subset R^n$ such that:

$$f(x^*) = \sup_{x \in D} f(x).$$

A local maximum is found when $\hat{x}$ satisfies $f(x) < f(\hat{x})$ for all $x \in D$ for some neighborhood defined by $0 < \|x - \hat{x}\| \leq \delta$, some δ.

Optimization Problem Definition Without Derivatives

Without requiring differentiability, the basic optimization problem is given by:
Find $x^* \in D$ for some domain $D \subset R^n$ such that:

$$f(x^*) = \sup_{x \in D} f(x).$$

A local maximum is found when $\hat{x}$ satisfies $f(x) < f(\hat{x})$ for all $x \in D$ for some neighborhood defined by $0 < \|x - \hat{x}\| \leq \delta$, some δ.

Stochastic Optimization With Derivatives [BR] (I of III)

Suppose that $f(.)$ is differentiable with derivative $g(.)$.

We can approximate the derivatives using finite differences in a random direction (like gradient ascent):

Step 1. Generate vector V_i randomly distributed over the unit sphere: $||V_i|| = 1$.

Step 2. Move in this random direction using:
$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_i) - f(x_i - c_i V_i)]/(2c_i)]\} V_i$

Here, α_i represents the step size and c_i represents a displacement for finite differencing

Stochastic Optimization With Derivatives [BR] (I of III)

Suppose that $f(.)$ is differentiable with derivative $g(.)$.

We can approximate the derivatives using finite differences in a random direction (like gradient ascent):

Step 1. Generate vector V_i randomly distributed over the unit sphere: $||V_i|| = 1$.

Step 2. Move in this random direction using:
$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_i) - f(x_i - c_i V_i)]/(2c_i)]\} V_i$

Here, α_i represents the step size and c_i represents a displacement for finite differencing

Stochastic Optimization With Derivatives [BR] (I of III)

Suppose that $f(.)$ is differentiable with derivative $g(.)$.

We can approximate the derivatives using finite differences in a random direction (like gradient ascent):

Step 1. Generate vector V_i randomly distributed over the unit sphere: $||V_i|| = 1$.

Step 2. Move in this random direction using:

$$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_i) - f(x_i - c_i V_i)]/(2c_i)\} V_i$$

Here, α_i represents the step size and c_i represents a displacement for finite differencing

Stochastic Optimization With Derivatives [BR] (I of III)

Suppose that $f(.)$ is differentiable with derivative $g(.)$.

We can approximate the derivatives using finite differences in a random direction (like gradient ascent):

Step 1. Generate vector V_i randomly distributed over the unit sphere: $||V_i|| = 1$.

Step 2. Move in this random direction using:
$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_i) - f(x_i - c_i V_i)]/(2c_i)]\} V_i$

Here, α_i represents the step size and c_i represents a displacement for finite differencing

Stochastic Optimization With Derivatives [BR] (II of III)

A modified approach will move to the most promising point:

Step 1. Generate N vectors V_{iq} randomly distributed over the unit sphere: $||V_{iq}|| = 1$.

Step 2. Choose the k that maximizes $f(x_i + c_i V_{ik})$.

Step 3. Move to this point:

$$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_{ik}) - f(x_i - c_i V_{ik})]/(2c_i)]\} V_{ik}$$

Stochastic Optimization With Derivatives [BR] (II of III)

A modified approach will move to the most promising point:

Step 1. Generate N vectors V_{iq} randomly distributed over the unit sphere: $||V_{iq}|| = 1$.

Step 2. Choose the k that maximizes $f(x_i + c_i V_{ik})$.

Step 3. Move to this point:

$$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_{ik}) - f(x_i - c_i V_{ik})]/(2c_i)]\} V_{ik}$$

Stochastic Optimization With Derivatives [BR] (II of III)

A modified approach will move to the most promising point:

Step 1. Generate N vectors V_{iq} randomly distributed over the unit sphere: $||V_{iq}|| = 1$.

Step 2. Choose the k that maximizes $f(x_i + c_i V_{ik})$.

Step 3. Move to this point:

$$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_{ik}) - f(x_i - c_i V_{ik})]/(2c_i)]\} V_{ik}$$

Stochastic Optimization With Derivatives [BR] (II of III)

A modified approach will move to the most promising point:

Step 1. Generate N vectors V_{iq} randomly distributed over the unit sphere: $||V_{iq}|| = 1$.

Step 2. Choose the k that maximizes $f(x_i + c_i V_{ik})$.

Step 3. Move to this point:

$$x_{i+1} = x_i + \alpha_i \{[f(x_i + c_i V_{ik}) - f(x_i - c_i V_{ik})]/(2c_i)]\} V_{ik}$$

Most Common Global Optimization Approach [BR] (III of III)

A very promising global + local approach will combine approaches:

Step 1. Select N starting points $x_i \in D$ based on some distribution over D

Step 2. Run a local optimization algorithm from each x_i to reach $\hat{x}_i$.

Step 3. Choose x^* as the $\hat{x}_i$ with the largest $f(\hat{x}_i)$.

How do we pick an appropriate distribution?

Most Common Global Optimization Approach [BR] (III of III)

A very promising global + local approach will combine approaches:

Step 1. Select N starting points $x_i \in D$ based on some distribution over D

Step 2. Run a local optimization algorithm from each x_i to reach $\hat{x}_i$.

Step 3. Choose x^* as the $\hat{x}_i$ with the largest $f(\hat{x}_i)$.

How do we pick an appropriate distribution?

Most Common Global Optimization Approach [BR] (III of III)

A very promising global + local approach will combine approaches:

Step 1. Select N starting points $x_i \in D$ based on some distribution over D

Step 2. Run a local optimization algorithm from each x_i to reach $\hat{x}_i$.

Step 3. Choose x^* as the $\hat{x}_i$ with the largest $f(\hat{x}_i)$.

How do we pick an appropriate distribution?

Most Common Global Optimization Approach [BR] (III of III)

A very promising global + local approach will combine approaches:

Step 1. Select N starting points $x_i \in D$ based on some distribution over D

Step 2. Run a local optimization algorithm from each x_i to reach $\hat{x}_i$.

Step 3. Choose x^* as the $\hat{x}_i$ with the largest $f(\hat{x}_i)$.

How do we pick an appropriate distribution?

Most Common Global Optimization Approach [BR] (III of III)

A very promising global + local approach will combine approaches:

Step 1. Select N starting points $x_i \in D$ based on some distribution over D

Step 2. Run a local optimization algorithm from each x_i to reach $\hat{x}_i$.

Step 3. Choose x^* as the $\hat{x}_i$ with the largest $f(\hat{x}_i)$.

How do we pick an appropriate distribution?

A Monte-Carlo Approach to Picking the Distribution

Ripley proves the following result on page 180.

Suppose that $f(.)$ is continuous with a unique global maximum at x^* and D is compact.

We have that:

$$x^* = \lim_{\lambda \to \infty} \frac{\int_D x \exp[\lambda f(x)] dx}{\int_D \exp[\lambda f(x)] dx}$$

We will follow Ross in taking a discrete approach to the problem (Section 10.4).

A Monte-Carlo Approach to Picking the Distribution

Ripley proves the following result on page 180.

Suppose that $f(.)$ is continuous with a unique global maximum at x^* and D is compact.

We have that:

$$x^* = \lim_{\lambda \to \infty} \frac{\int_D x \exp[\lambda f(x)] dx}{\int_D \exp[\lambda f(x)] dx}$$

We will follow Ross in taking a discrete approach to the problem (Section 10.4).

A Monte-Carlo Approach to Picking the Distribution

Ripley proves the following result on page 180.

Suppose that $f(.)$ is continuous with a unique global maximum at x^* and D is compact.

We have that:

$$x^* = \lim_{\lambda \to \infty} \frac{\int_D x \exp[\lambda f(x)] dx}{\int_D \exp[\lambda f(x)] dx}$$

We will follow Ross in taking a discrete approach to the problem (Section 10.4).

A Monte-Carlo Approach to Picking the Distribution

Ripley proves the following result on page 180.

Suppose that $f(.)$ is continuous with a unique global maximum at x^* and D is compact.

We have that:

$$x^* = \lim_{\lambda \to \infty} \frac{\int_D x \exp[\lambda f(x)] dx}{\int_D \exp[\lambda f(x)] dx}$$

We will follow Ross in taking a discrete approach to the problem (Section 10.4).

Outline

Optimization Over a Finite Number of Points

Let $\mathcal{A}$ be a finite set of vectors.

Let $f(.)$ be the function that we want to optimize.

We want to find at-least one $x^* \in \mathcal{M}$ where we have that:

$$\mathcal{M} = \{x \in \mathcal{A} \quad \text{such that:} \quad f(x) = f^*\}$$

and:

$$f^* = \max_{x \in \mathcal{A}} f(x).$$

Optimization Over a Finite Number of Points

Let $\mathcal{A}$ be a finite set of vectors.
Let $f(.)$ be the function that we want to optimize.

We want to find at-least one $x^* \in \mathcal{M}$ where we have that:

$$\mathcal{M} = \{x \in \mathcal{A} \quad \text{such that:} \quad f(x) = f^*\}$$

and:

$$f^* = \max_{x \in \mathcal{A}} f(x).$$

Optimization Over a Finite Number of Points

Let $\mathcal{A}$ be a finite set of vectors.
Let $f(.)$ be the function that we want to optimize.

We want to find at-least one $x^* \in \mathcal{M}$ where we have that:

$$\mathcal{M} = \{x \in \mathcal{A} \quad \text{such that:} \quad f(x) = f^*\}$$

and:

$$f^* = \max_{x \in \mathcal{A}} f(x).$$

A Discrete Distribution that will work

For $\lambda > 0$, define the probability mass function given by

$$p_\lambda(x) = \frac{e^{\lambda f(x)}}{\sum_{x \in \mathcal{A}} e^{\lambda f(x)}}$$

After multiplying both the numerator and the denominator by $e^{-\lambda f^*}$, and letting $|\mathcal{M}$ be the number of element in $\mathcal{M}$, we get:

$$p_\lambda(x) = \frac{e^{\lambda(f(x)-f^*)}}{|\mathcal{M}| + \sum_{x \notin \mathcal{M}} e^{\lambda(f(x)-f^*)}}$$

Please note that $f(x) - f^* < 0$ for $x \notin \mathcal{M}$. Thus, as $\lambda \to \infty$, we get

$$p_\lambda(x) \to \frac{\delta(x, \mathcal{M})}{|\mathcal{M}|}.$$

where $\delta(x, \mathcal{M}) = 1$ if $x \in \mathcal{M}$ and 0 otherwise.

A Discrete Distribution that will work

For $\lambda > 0$, define the probability mass function given by

$$p_\lambda(x) = \frac{e^{\lambda f(x)}}{\sum_{x \in \mathcal{A}} e^{\lambda f(x)}}$$

After multiplying both the numerator and the denominator by $e^{-\lambda f^*}$, and letting $|\mathcal{M}$ be the number of element in $\mathcal{M}$, we get:

$$p_\lambda(x) = \frac{e^{\lambda (f(x) - f^*)}}{|\mathcal{M}| + \sum_{x \notin \mathcal{M}} e^{\lambda (f(x) - f^*)}}$$

Please note that $f(x) - f^* < 0$ for $x \notin \mathcal{M}$. Thus, as $\lambda \to \infty$, we get

$$p_\lambda(x) \to \frac{\delta(x, \mathcal{M})}{|\mathcal{M}|},$$

where $\delta(x, \mathcal{M}) = 1$ if $x \in \mathcal{M}$ and 0 otherwise.

A Discrete Distribution that will work

For $\lambda > 0$, define the probability mass function given by

$$p_\lambda(x) = \frac{e^{\lambda f(x)}}{\sum_{x \in \mathcal{A}} e^{\lambda f(x)}}$$

After multiplying both the numerator and the denominator by $e^{-\lambda f^*}$, and letting $|\mathcal{M}$ be the number of element in $\mathcal{M}$, we get:

$$p_\lambda(x) = \frac{e^{\lambda (f(x) - f^*)}}{|\mathcal{M}| + \sum_{x \notin \mathcal{M}} e^{\lambda (f(x) - f^*)}}$$

Please note that $f(x) - f^* < 0$ for $x \notin \mathcal{M}$. Thus, as $\lambda \to \infty$, we get

$$p_\lambda(x) \to \frac{\delta(x, \mathcal{M})}{|\mathcal{M}|}.$$

where $\delta(x, \mathcal{M}) = 1$ if $x \in \mathcal{M}$ and 0 otherwise.

Outline

Generate a Markov Chain

Define a neighborhood system.

For example, two vectors are neighbors if they differ by 1 in one coordinate. Eg: $x = (x_1, x_2)$ has four neighbors: $(x_1, x_2 \pm 1)$, $(x_1 \pm 1, x_2)$.

Define a Markov Chain on how to move from one vector to its neighbors. Let $|N(x)|$ denote the number of neighbors of x.

At step n of a simulated Markov Chain, we move from x, to a neighbor y with probability

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\}$$

Else, we stay at x with probability $1 - p$.

Generate a Markov Chain

Define a neighborhood system.

For example, two vectors are neighbors if they differ by 1 in one coordinate. Eg: $x = (x_1, x_2)$ has four neighbors: $(x_1, x_2 \pm 1)$, $(x_1 \pm 1, x_2)$.

Define a Markov Chain on how to move from one vector to its neighbors. Let $|N(x)|$ denote the number of neighbors of x.

At step n of a simulated Markov Chain, we move from x, to a neighbor y with probability

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\}$$

Else, we stay at x with probability $1 - p$.

Generate a Markov Chain

Define a neighborhood system.

For example, two vectors are neighbors if they differ by 1 in one coordinate. Eg: $x = (x_1, x_2)$ has four neighbors: $(x_1, x_2 \pm 1)$, $(x_1 \pm 1, x_2)$.

Define a Markov Chain on how to move from one vector to its neighbors. Let $|N(x)|$ denote the number of neighbors of x.

At step n of a simulated Markov Chain, we move from x, to a neighbor y with probability

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\}$$

Else, we stay at x with probability $1 - p$.

Generate a Markov Chain

Define a neighborhood system.
For example, two vectors are neighbors if they differ by 1 in one coordinate. Eg: $x = (x_1, x_2)$ has four neighbors: $(x_1, x_2 \pm 1)$, $(x_1 \pm 1, x_2)$.

Define a Markov Chain on how to move from one vector to its neighbors. Let $|N(x)|$ denote the number of neighbors of x.

At step n of a simulated Markov Chain, we move from x, to a neighbor y with probability

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\}$$

Else, we stay at x with probability $1 - p$.

Generate a Markov Chain

Define a neighborhood system.

For example, two vectors are neighbors if they differ by 1 in one coordinate. Eg: $x = (x_1, x_2)$ has four neighbors: $(x_1, x_2 \pm 1)$, $(x_1 \pm 1, x_2)$.

Define a Markov Chain on how to move from one vector to its neighbors. Let $|N(x)|$ denote the number of neighbors of x.

At step n of a simulated Markov Chain, we move from x, to a neighbor y with probability

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\}$$

Else, we stay at x with probability $1 - p$.

Generate a Markov Chain

Define a neighborhood system.
For example, two vectors are neighbors if they differ by 1 in one coordinate. Eg: $x = (x_1, x_2)$ has four neighbors: $(x_1, x_2 \pm 1)$, $(x_1 \pm 1, x_2)$.

Define a Markov Chain on how to move from one vector to its neighbors. Let $|N(x)|$ denote the number of neighbors of x.

At step n of a simulated Markov Chain, we move from x, to a neighbor y with probability

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\}$$

Else, we stay at x with probability $1 - p$.

Simulated Annealing Algorithm

For convergence, we need $\lambda_n = C \log(1+n)$ with $C > 0$.

The algorithm is summarized into two steps:

Step 1. Generate Markov Chain states $X_1, X_2, \ldots, X_m$.

Step 2. Estimate f_M^* and x_M^* using:

$$f_M^* = \max\{f(X_1), f(X_2), \ldots, f(X_m)\}$$

and

$$x_M^* = X_i \quad \text{for the maximum point:} \quad f(X_i) = f_M^*.$$

Simulated Annealing Algorithm

For convergence, we need $\lambda_n = C \log(1+n)$ with $C > 0$.
The algorithm is summarized into two steps:

Step 1. Generate Markov Chain states $X_1, X_2, \ldots, X_m$.

Step 2. Estimate f_M^* and x_M^* using:

$$f_M^* = \max\{f(X_1), f(X_2), \ldots, f(X_m)\}$$

and

$$x_M^* = X_i \quad \text{for the maximum point:} \quad f(X_i) = f_M^*.$$

Simulated Annealing Algorithm

For convergence, we need $\lambda_n = C \log(1+n)$ with $C > 0$.
The algorithm is summarized into two steps:

Step 1. Generate Markov Chain states $X_1, X_2, \ldots, X_m$.

Step 2. Estimate f_M^* and x_M^* using:

$$f_M^* = \max\{f(X_1), f(X_2), \ldots, f(X_m)\}$$

and

$$x_M^* = X_i \quad \text{for the maximum point:} \quad f(X_i) = f_M^*.$$

Simulated Annealing Algorithm

For convergence, we need $\lambda_n = C \log(1+n)$ with $C > 0$.
The algorithm is summarized into two steps:

Step 1. Generate Markov Chain states $X_1, X_2, \ldots, X_m$.

Step 2. Estimate f_M^* and x_M^* using:

$$f_M^* = \max\{f(X_1), f(X_2), \ldots, f(X_m)\}$$

and

$$x_M^* = X_i \quad \text{for the maximum point:} \quad f(X_i) = f_M^*.$$

Simulated Annealing Algorithm

For convergence, we need $\lambda_n = C \log(1+n)$ with $C > 0$.
The algorithm is summarized into two steps:

Step 1. Generate Markov Chain states $X_1, X_2, \ldots, X_m$.

Step 2. Estimate f_M^* and x_M^* using:

$$f_M^* = \max\{f(X_1), f(X_2), \ldots, f(X_m)\}$$

and

$$x_M^* = X_i \quad \text{for the maximum point:} \quad f(X_i) = f_M^*.$$

Outline

Combinatorial Optimization: TSP

For the Traveling Salesman Problem (TSP), we want to find the optimal city ordering, so that the reward from visiting all the cities is maximized. Assume that we will return to the originating city.

This is a combinatorial optimization problem which can only be solved after we evaluate all possibilities. Simulated annealing provides for an approximate solution method.

Let $0, 1, 2, \ldots, r$ denote all the possible cities.
Then, a permutation $x_1, \ldots, x_r$ of $1, 2, \ldots, r$ represents a possible tour. Then, define the cost function to maximize is the sum of all rewards

$$f(x) = \sum_{i=1}^{r} g(x_{i-1}, x_i), \qquad x_0 = 0.$$

Combinatorial Optimization: TSP

For the Traveling Salesman Problem (TSP), we want to find the optimal city ordering, so that the reward from visiting all the cities is maximized. Assume that we will return to the originating city.

This is a combinatorial optimization problem which can only be solved after we evaluate all possibilities. Simulated annealing provides for an approximate solution method.

Let $0, 1, 2, \ldots, r$ denote all the possible cities.
Then, a permutation $x_1, \ldots, x_r$ of $1, 2, \ldots, r$ represents a possible tour. Then, define the cost function to maximize is the sum of all rewards

$$f(x) = \sum_{i=1}^{r} g(x_{i-1}, x_i), \qquad x_0 = 0.$$

Combinatorial Optimization: TSP

For the Traveling Salesman Problem (TSP), we want to find the optimal city ordering, so that the reward from visiting all the cities is maximized. Assume that we will return to the originating city.

This is a combinatorial optimization problem which can only be solved after we evaluate all possibilities. Simulated annealing provides for an approximate solution method.

Let $0, 1, 2, \ldots, r$ denote all the possible cities.
Then, a permutation $x_1, \ldots, x_r$ of $1, 2, \ldots, r$ represents a possible tour. Then, define the cost function to maximize is the sum of all rewards

$$f(x) = \sum_{i=1}^{r} g(x_{i-1}, x_i), \qquad x_0 = 0.$$

Combinatorial Optimization: TSP

For the Traveling Salesman Problem (TSP), we want to find the optimal city ordering, so that the reward from visiting all the cities is maximized. Assume that we will return to the originating city.

This is a combinatorial optimization problem which can only be solved after we evaluate all possibilities. Simulated annealing provides for an approximate solution method.

Let $0, 1, 2, \ldots, r$ denote all the possible cities.

Then, a permutation $x_1, \ldots, x_r$ of $1, 2, \ldots, r$ represents a possible tour. Then, define the cost function to maximize is the sum of all rewards

$$f(x) = \sum_{i=1}^{r} g(x_{i-1}, x_i), \qquad x_0 = 0.$$

Combinatorial Optimization: TSP

For the Traveling Salesman Problem (TSP), we want to find the optimal city ordering, so that the reward from visiting all the cities is maximized. Assume that we will return to the originating city.

This is a combinatorial optimization problem which can only be solved after we evaluate all possibilities. Simulated annealing provides for an approximate solution method.

Let $0, 1, 2, \ldots, r$ denote all the possible cities.
Then, a permutation $x_1, \ldots, x_r$ of $1, 2, \ldots, r$ represents a possible tour. Then, define the cost function to maximize is the sum of all rewards

$$f(x) = \sum_{i=1}^{r} g(x_{i-1}, x_i), \qquad x_0 = 0.$$

Working with a TSP neighborhood

A neighbor is defined by the swap operation.

Thus, $1, 2, 3$ is a neighbor of $3, 2, 1$ since 1 and 3 have switched positions.

To generate a neighbor, simply pick a pair in two steps:

Step 1. Pick a I from $1, \ldots, r$.

Step 2. Pick a J from $1, \ldots, I-1, I+1, \ldots, r$, avoiding I.

We then exchange the I-th with the J-th city to generate a new, neighboring tour.

Here, we would not have to count the number of neighbors in order to build the probability distributions.

To count them, note that we have $r \times (r-1)/(1 \times 2)$ or $\binom{r}{2}$ possible neighbors.

Working with a TSP neighborhood

A neighbor is defined by the swap operation.
Thus, $1, 2, 3$ is a neighbor of $3, 2, 1$ since 1 and 3 have switched positions.

To generate a neighbor, simply pick a pair in two steps:

Step 1. Pick a I from $1, \ldots, r$.

Step 2. Pick a J from $1, \ldots, I-1, I+1, \ldots, r$, avoiding I.
We then exchange the I-th with the J-th city to generate a new, neighboring tour.

Here, we would not have to count the number of neighbors in order to build the probability distributions.

To count them, note that we have $r \times (r-1)/(1 \times 2)$ or $\binom{r}{2}$ possible neighbors.

Working with a TSP neighborhood

A neighbor is defined by the swap operation.
Thus, $1, 2, 3$ is a neighbor of $3, 2, 1$ since 1 and 3 have switched positions.

To generate a neighbor, simply pick a pair in two steps:

Step 1. Pick a I from $1, \ldots, r$.

Step 2. Pick a J from $1, \ldots, I-1, I+1, \ldots, r$, avoiding I.
We then exchange the I-th with the J-th city to generate a new, neighboring tour.

Here, we would not have to count the number of neighbors in order to build the probability distributions.

To count them, note that we have $r \times (r-1)/(1 \times 2)$ or $\binom{r}{2}$ possible neighbors.

Working with a TSP neighborhood

A neighbor is defined by the swap operation.
Thus, $1, 2, 3$ is a neighbor of $3, 2, 1$ since 1 and 3 have switched positions.

To generate a neighbor, simply pick a pair in two steps:

Step 1. Pick a I from $1, \ldots, r$.

Step 2. Pick a J from $1, \ldots, I-1, I+1, \ldots, r$, avoiding I.

We then exchange the I-th with the J-th city to generate a new, neighboring tour.

Here, we would not have to count the number of neighbors in order to build the probability distributions.

To count them, note that we have $r \times (r-1)/(1 \times 2)$ or $\binom{r}{2}$ possible neighbors.

Working with a TSP neighborhood

A neighbor is defined by the swap operation.
Thus, $1, 2, 3$ is a neighbor of $3, 2, 1$ since 1 and 3 have switched positions.

To generate a neighbor, simply pick a pair in two steps:

Step 1. Pick a I from $1, \ldots, r$.

Step 2. Pick a J from $1, \ldots, I-1, I+1, \ldots, r$, avoiding I.
We then exchange the I-th with the J-th city to generate a new, neighboring tour.

Here, we would not have to count the number of neighbors in order to build the probability distributions.

To count them, note that we have $r \times (r-1)/(1 \times 2)$ or $\binom{r}{2}$ possible neighbors.

Working with a TSP neighborhood

A neighbor is defined by the swap operation.
Thus, $1, 2, 3$ is a neighbor of $3, 2, 1$ since 1 and 3 have switched positions.

To generate a neighbor, simply pick a pair in two steps:

Step 1. Pick a I from $1, \ldots, r$.

Step 2. Pick a J from $1, \ldots, I-1, I+1, \ldots, r$, avoiding I.
We then exchange the I-th with the J-th city to generate a new, neighboring tour.

Here, we would not have to count the number of neighbors in order to build the probability distributions.

To count them, note that we have $r \times (r-1)/(1 \times 2)$ or $\binom{r}{2}$ possible neighbors.

Working with a TSP neighborhood

A neighbor is defined by the swap operation.
Thus, $1, 2, 3$ is a neighbor of $3, 2, 1$ since 1 and 3 have switched positions.

To generate a neighbor, simply pick a pair in two steps:

Step 1. Pick a I from $1, \ldots, r$.

Step 2. Pick a J from $1, \ldots, I-1, I+1, \ldots, r$, avoiding I.
We then exchange the I-th with the J-th city to generate a new, neighboring tour.

Here, we would not have to count the number of neighbors in order to build the probability distributions.

To count them, note that we have $r \times (r-1)/(1 \times 2)$ or $\binom{r}{2}$ possible neighbors.

TSP Solution

Use $\lambda_n = \log(1+n)$ to get:

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\} = \min\left\{1, (n+1)^{f(y)-f(x)}\right\}.$$

If $f(y) > f(x)$, **then** $p = 1$ and we **Set** $X_{n+1} = y$.

Else

Set $X_{n+1} = y$ with probability $p = (n+1)^{f(y)-f(x)}$.

Else Set $X_{n+1} = x$ with probability $1-p$.

TSP Solution

Use $\lambda_n = \log(1+n)$ to get:

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\} = \min\left\{1, (n+1)^{f(y)-f(x)}\right\}.$$

If $f(y) > f(x)$, **then** $p = 1$ and we **Set** $X_{n+1} = y$.
Else

Set $X_{n+1} = y$ with probability $p = (n+1)^{f(y)-f(x)}$.
Else Set $X_{n+1} = x$ with probability $1-p$.

TSP Solution

Use $\lambda_n = \log(1+n)$ to get:

$$p = \min\left\{1, \frac{\exp\left(\lambda_n f(y)\right)/|N(y)|}{\exp\left(\lambda_n f(x)\right)/|N(x)|}\right\} = \min\left\{1, (n+1)^{f(y)-f(x)}\right\}.$$

If $f(y) > f(x)$, **then** $p = 1$ and we **Set** $X_{n+1} = y$.
Else

Set $X_{n+1} = y$ with probability $p = (n+1)^{f(y)-f(x)}$.
Else Set $X_{n+1} = x$ with probability $1-p$.

Remaining Questions

- How do we generate a discrete random number?
- How do we generate a continuous random number?
- How do we relate the Markov Chain to the multivariate PDF?

Remaining Questions

- How do we generate a discrete random number?
- How do we generate a continuous random number?
- How do we relate the Markov Chain to the multivariate PDF?

Remaining Questions

- How do we generate a discrete random number?
- How do we generate a continuous random number?
- How do we relate the Markov Chain to the multivariate PDF?

Outline

Pseudorandom Number Generation

The simplest uniformly distributed random number generator uses:

$$X_n = (aX_{n-1} + c) \bmod m$$

where:

- X_0 denotes an initial value.
- m determines the period of repetition.
- a multiplicative constant to be fixed.
- c additive constant that can also be zero.

Ripley recommends the widely used (page 46):

$$X_n = (69069X_{n-1} + 1) \bmod 2^{32}, \quad U_n = 2^{-32}X_i \quad \text{is} \quad [0, 1)$$

Pseudorandom Number Generation

The simplest uniformly distributed random number generator uses:

$$X_n = (aX_{n-1} + c) \bmod m$$

where:

- X_0 denotes an initial value.
- m determines the period of repetition.
- a multiplicative constant to be fixed.
- c additive constant that can also be zero.

Ripley recommends the widely used (page 46):

$$X_n = (69069X_{n-1} + 1) \bmod 2^{32}, \quad U_n = 2^{-32}X_i \quad \text{is} \quad [0, 1)$$

Discrete Random Number Generation

We want to generate the discrete variable X:

$$P\{X = x_j\} = p_j, \quad \sum_j p_j = 1$$

Generate a uniformly distributed RV: $U\ (0,1)$.

$$X = \begin{cases} x_0 & \text{if} \quad U < p_0 \\ x_1 & \text{if} \quad p_0 \le U < p_0 + p_1 \\ \vdots & \\ x_j & \text{if} \quad \sum_{i=1}^{j-1} p_i \le U < \sum_{i=1}^{j} p_i \\ \vdots & \end{cases}$$

Discrete Random Number Generation

We want to generate the discrete variable X:

$$P\{X = x_j\} = p_j, \quad \sum_j p_j = 1$$

Generate a uniformly distributed RV: $U\ (0, 1)$.

$$X = \begin{cases} x_0 & \text{if} \quad U < p_0 \\ x_1 & \text{if} \quad p_0 \leq U < p_0 + p_1 \\ \vdots & \\ x_j & \text{if} \quad \sum_{i=1}^{j-1} p_i \leq U < \sum_{i=1}^{j} p_i \\ \vdots & \end{cases}$$

Discrete Random Number Generation

We want to generate the discrete variable X:

$$P\{X = x_j\} = p_j, \quad \sum_j p_j = 1$$

Generate a uniformly distributed RV: $U\ (0, 1)$.

$$X = \begin{cases} x_0 & \text{if} \quad U < p_0 \\ x_1 & \text{if} \quad p_0 \leq U < p_0 + p_1 \\ \vdots & \\ x_j & \text{if} \quad \sum_{i=1}^{j-1} p_i \leq U < \sum_{i=1}^{j} p_i \\ \vdots & \end{cases}$$

Box-Muller for Continuous Normal Random Variables

Ripley gives the following "exact" method for $N(0,1)$ [page 54]:

Step 1. Generate U_1. Set $\Theta = 2\pi U_1$.

Step 2. Generate U_2. Set $E = -\ln U_2$, $R = \sqrt{2E}$.

Step 3. Set $X = R\cos\Theta$, $Y = R\sin\Theta$.

The algorithm generates two normally distributed random numbers. If we only need one, then we do not need to compute Y.

Box-Muller for Continuous Normal Random Variables

Ripley gives the following "exact" method for $N(0,1)$ [page 54]:

Step 1. Generate U_1. Set $\Theta = 2\pi U_1$.

Step 2. Generate U_2. Set $E = -\ln U_2$, $R = \sqrt{2E}$.

Step 3. Set $X = R\cos\Theta$, $Y = R\sin\Theta$.

The algorithm generates two normally distributed random numbers. If we only need one, then we do not need to compute Y.

Box-Muller for Continuous Normal Random Variables

Ripley gives the following "exact" method for $N(0,1)$ [page 54]:

Step 1. Generate U_1. Set $\Theta = 2\pi U_1$.

Step 2. Generate U_2. Set $E = -\ln U_2$, $R = \sqrt{2E}$.

Step 3. Set $X = R\cos\Theta$, $Y = R\sin\Theta$.

The algorithm generates two normally distributed random numbers. If we only need one, then we do not need to compute Y.

Box-Muller for Continuous Normal Random Variables

Ripley gives the following "exact" method for $N(0,1)$ [page 54]:

Step 1. Generate U_1. Set $\Theta = 2\pi U_1$.

Step 2. Generate U_2. Set $E = -\ln U_2$, $R = \sqrt{2E}$.

Step 3. Set $X = R\cos\Theta$, $Y = R\sin\Theta$.

The algorithm generates two normally distributed random numbers. If we only need one, then we do not need to compute Y.

Continuous Random Number Generation [SR]

A general method for generating continuous random variables is based on the **rejection method**.

Suppose that we have a method for generating random variables from density $g(.)$. We want a method that has density $f(.)$.

Here is the basic algorithm:

Step 1. Generate Y based on $g(.)$

Step 2. Generate a uniformly distributed U

Step 3. **If** $U \leq f(Y)/(cg(Y))$ Set $X = Y$, **(accept)**.

Else repeat step 1, **(reject)**.

We need to select c so that:

$$\frac{f(y)}{g(y)} \leq c \quad \text{for all} \quad y$$

Continuous Random Number Generation [SR]

A general method for generating continuous random variables is based on the **rejection method**.
Suppose that we have a method for generating random variables from density $g(.)$. We want a method that has density $f(.)$.

Here is the basic algorithm:

Step 1. Generate Y based on $g(.)$

Step 2. Generate a uniformly distributed U

Step 3. **If** $U \leq f(Y)/(cg(Y))$ Set $X = Y$, **(accept)**.
Else repeat step 1, **(reject)**.

We need to select c so that:

$$\frac{f(y)}{g(y)} \leq c \quad \text{for all} \quad y$$

Continuous Random Number Generation [SR]

A general method for generating continuous random variables is based on the **rejection method**.
Suppose that we have a method for generating random variables from density $g(.)$. We want a method that has density $f(.)$.

Here is the basic algorithm:

Step 1. Generate Y based on $g(.)$

Step 2. Generate a uniformly distributed U

Step 3. **If** $U \leq f(Y)/(cg(Y))$ Set $X = Y$, **(accept)**.
Else repeat step 1, **(reject)**.

We need to select c so that:

$$\frac{f(y)}{g(y)} \leq c \quad \text{for all} \quad y$$

Continuous Random Number Generation [SR]

A general method for generating continuous random variables is based on the **rejection method**.
Suppose that we have a method for generating random variables from density $g(.)$. We want a method that has density $f(.)$.

Here is the basic algorithm:

Step 1. Generate Y based on $g(.)$

Step 2. Generate a uniformly distributed U

Step 3. **If** $U \leq f(Y)/(cg(Y))$ Set $X = Y$, **(accept)**.
Else repeat step 1, **(reject)**.

We need to select c so that:

$$\frac{f(y)}{g(y)} \leq c \quad \text{for all} \quad y$$

Continuous Random Number Generation [SR]

A general method for generating continuous random variables is based on the **rejection method**.
Suppose that we have a method for generating random variables from density $g(.)$. We want a method that has density $f(.)$.

Here is the basic algorithm:

Step 1. Generate Y based on $g(.)$

Step 2. Generate a uniformly distributed U

Step 3. **If** $U \leq f(Y)/(cg(Y))$ Set $X = Y$, **(accept)**.
Else repeat step 1, **(reject)**.

We need to select c so that:

$$\frac{f(y)}{g(y)} \leq c \quad \text{for all} \quad y$$

Continuous Random Number Generation [SR]

A general method for generating continuous random variables is based on the **rejection method**.
Suppose that we have a method for generating random variables from density $g(.)$. We want a method that has density $f(.)$.

Here is the basic algorithm:

Step 1. Generate Y based on $g(.)$

Step 2. Generate a uniformly distributed U

Step 3. **If** $U \leq f(Y)/(cg(Y))$ Set $X = Y$, **(accept)**.

Else repeat step 1, **(reject)**.

We need to select c so that:

$$\frac{f(y)}{g(y)} \leq c \quad \text{for all} \quad y$$

Continuous Random Number Generation [SR]

A general method for generating continuous random variables is based on the **rejection method**.
Suppose that we have a method for generating random variables from density $g(.)$. We want a method that has density $f(.)$.

Here is the basic algorithm:

Step 1. Generate Y based on $g(.)$

Step 2. Generate a uniformly distributed U

Step 3. **If** $U \leq f(Y)/(cg(Y))$ Set $X = Y$, **(accept)**.
Else repeat step 1, **(reject)**.

We need to select c so that:

$$\frac{f(y)}{g(y)} \leq c \quad \text{for all} \quad y$$

Mapping M-D Arrays to 1-D Arrays

Note that we can come up with simple ways to map discrete variables of two or more dimensions to the integers: $1, 2, \ldots, R$.

Example: Consider the simplest case of mapping a two-dimensional array of points: $A[i, j]$. Let A be of size $N \times M$. We can map every 2-D index (i, j) to a 1-D index using $r = N * (j - 1) + i$ (column-wise) and also set $R = N \cdot M$.

Also, we can map every 1-D index to 2-D using:

$$i = r \bmod N$$
$$j = (r - i)/N + 1.$$

This idea is easy to extend to any finite number of dimensions. Use `sub2ind(.)` and `ind2sub` in Matlab.

Mapping M-D Arrays to 1-D Arrays

Note that we can come up with simple ways to map discrete variables of two or more dimensions to the integers: $1, 2, \ldots, R$.

Example: Consider the simplest case of mapping a two-dimensional array of points: $A[i, j]$. Let A be of size $N \times M$. We can map every 2-D index (i, j) to a 1-D index using $r = N * (j - 1) + i$ (column-wise) and also set $R = N \cdot M$.

Also, we can map every 1-D index to 2-D using:

$$i = r \bmod N$$
$$j = (r - i)/N + 1.$$

This idea is easy to extend to any finite number of dimensions. Use `sub2ind(.)` and `ind2sub` in Matlab.

Mapping M-D Arrays to 1-D Arrays

Note that we can come up with simple ways to map discrete variables of two or more dimensions to the integers: $1, 2, \ldots, R$.

Example: Consider the simplest case of mapping a two-dimensional array of points: $A[i, j]$. Let A be of size $N \times M$. We can map every 2-D index (i, j) to a 1-D index using $r = N * (j - 1) + i$ (column-wise) and also set $R = N \cdot M$.

Also, we can map every 1-D index to 2-D using:

$$i = r \bmod N$$

$$j = (r - i)/N + 1.$$

This idea is easy to extend to any finite number of dimensions. Use `sub2ind(.)` and `ind2sub` in Matlab.

Mapping M-D Arrays to 1-D Arrays

Note that we can come up with simple ways to map discrete variables of two or more dimensions to the integers: $1, 2, \ldots, R$.

Example: Consider the simplest case of mapping a two-dimensional array of points: $A[i, j]$. Let A be of size $N \times M$. We can map every 2-D index (i, j) to a 1-D index using $r = N * (j - 1) + i$ (column-wise) and also set $R = N \cdot M$.

Also, we can map every 1-D index to 2-D using:

$$i = r \bmod N$$
$$j = (r - i)/N + 1.$$

This idea is easy to extend to any finite number of dimensions. Use `sub2ind(.)` and `ind2sub` in Matlab.

Mapping M-D Arrays to 1-D Arrays

Note that we can come up with simple ways to map discrete variables of two or more dimensions to the integers: $1, 2, \ldots, R$.

Example: Consider the simplest case of mapping a two-dimensional array of points: $A[i, j]$. Let A be of size $N \times M$. We can map every 2-D index (i, j) to a 1-D index using $r = N * (j - 1) + i$ (column-wise) and also set $R = N \cdot M$.

Also, we can map every 1-D index to 2-D using:

$$i = r \bmod N$$
$$j = (r - i)/N + 1.$$

This idea is easy to extend to any finite number of dimensions. Use `sub2ind(.)` and `ind2sub` in Matlab.

Mapping M-D Arrays to 1-D Arrays

Note that we can come up with simple ways to map discrete variables of two or more dimensions to the integers: $1, 2, \ldots, R$.

Example: Consider the simplest case of mapping a two-dimensional array of points: $A[i, j]$. Let A be of size $N \times M$. We can map every 2-D index (i, j) to a 1-D index using $r = N * (j - 1) + i$ (column-wise) and also set $R = N \cdot M$.

Also, we can map every 1-D index to 2-D using:

$$i = r \bmod N$$
$$j = (r - i)/N + 1.$$

This idea is easy to extend to any finite number of dimensions. Use `sub2ind(.)` and `ind2sub` in `Matlab`.

Mapping M-D Transitional Probabilities to 1-D

For the transition probabilities, we can make them a function of distance between the points.

Start from:

$$q(r_1, r_2) = q((i_1 j_1), (i_2, j_2))$$

where r_1, r_2 correspond to $(i_1 j_1)$, (i_2, j_2).

Then set the distance using:

$$q(r_1, r_2) = f\left(d = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2}\right) / C_{r_1}.$$

for some $f(.) > 0$ and C_{r_1} to be determined.

Mapping M-D Transitional Probabilities to 1-D

For the transition probabilities, we can make them a function of distance between the points.
Start from:

$$q(r_1,\, r_2) = q((i_1\, j_1),\ (i_2,\, j_2))$$

where $r_1,\, r_2$ correspond to $(i_1\, j_1),\ (i_2,\, j_2)$.

Then set the distance using:

$$q(r_1,\, r_2) = f\left(d = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2}\right) / C_{r_1}.$$

for some $f(.) > 0$ and C_{r_1} to be determined.

Mapping M-D Transitional Probabilities to 1-D

For the transition probabilities, we can make them a function of distance between the points.
Start from:

$$q(r_1,\, r_2) = q((i_1\, j_1),\ (i_2,\, j_2))$$

where $r_1,\, r_2$ correspond to $(i_1\, j_1),\ (i_2,\, j_2)$.

Then set the distance using:

$$q(r_1,\, r_2) = f\left(d = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2}\right) / C_{r_1}.$$

for some $f(.) > 0$ and C_{r_1} to be determined.

Mapping M-D Transitional Probabilities to 1-D (Contd)

To determine C_{r_1}, we note that for the transition probability to be valid, it must add up to 1 along each row:

$$\sum_{j=1}^{R} q_{ij} = 1, \quad i = 1, 2, 3, \ldots, R.$$

Thus, starting from any $f(.) > 0$, we need to set all the C_{r_1} using: $C_{r_1} = \sum_{r_2=1}^{r_2=R} f(d(r_1, r_2)), \quad r_1 = 1, \ldots, R.$

For $f(.)$, we can use $f(d) = \exp(-d^2)$. However, it is very common to simply consider probabilities on the neighbors and set the rest of them to zero.

Mapping M-D Transitional Probabilities to 1-D (Contd)

To determine C_{r_1}, we note that for the transition probability to be valid, it must add up to 1 along each row:

$$\sum_{j=1}^{R} q_{ij} = 1, \quad i = 1, 2, 3, \ldots, R.$$

Thus, starting from any $f(.) > 0$, we need to set all the C_{r_1} using: $C_{r_1} = \sum_{r_2=1}^{r_2=R} f(d(r_1, r_2)), \quad r_1 = 1, \ldots, R.$

For $f(.)$, we can use $f(d) = \exp(-d^2)$. However, it is very common to simply consider probabilities on the neighbors and set the rest of them to zero.

Mapping M-D Transitional Probabilities to 1-D (Contd)

To determine C_{r_1}, we note that for the transition probability to be valid, it must add up to 1 along each row:

$$\sum_{j=1}^{R} q_{ij} = 1, \quad i = 1, 2, 3, \ldots, R.$$

Thus, starting from any $f(.) > 0$, we need to set all the C_{r_1} using: $C_{r_1} = \sum_{r_2=1}^{r_2=R} f(d(r_1, r_2)), \quad r_1 = 1, \ldots, R.$

For $f(.)$, we can use $f(d) = \exp(-d^2)$. However, it is very common to simply consider probabilities on the neighbors and set the rest of them to zero.

Mapping M-D Transitional Probabilities to 1-D (Contd)

To determine C_{r_1}, we note that for the transition probability to be valid, it must add up to 1 along each row:

$$\sum_{j=1}^{R} q_{ij} = 1, \quad i = 1, 2, 3, \ldots, R.$$

Thus, starting from any $f(.) > 0$, we need to set all the C_{r_1} using: $C_{r_1} = \sum_{r_2=1}^{r_2=R} f(d(r_1, r_2)), \quad r_1 = 1, \ldots, R.$

For $f(.)$, we can use $f(d) = \exp(-d^2)$. However, it is very common to simply consider probabilities on the neighbors and set the rest of them to zero.

Mapping M-D Transitional Probabilities to 1-D (Contd)

How about using discrete arrays for continuous random variables?

We can sample using:

$$A_d[i, j] = A_a(i\delta x, j\delta y)$$

where δx, δy can be determined based on the level of accuracy that is required.

Thus, after the algorithm terminates, the accuracy will be at-most δx in the first direction and δy in the second direction.

Mapping M-D Transitional Probabilities to 1-D (Contd)

How about using discrete arrays for continuous random variables?

We can sample using:

$$A_d[i, j] = A_a(i\delta x, j\delta y)$$

where δx, δy can be determined based on the level of accuracy that is required.

Thus, after the algorithm terminates, the accuracy will be at-most δx in the first direction and δy in the second direction.

Mapping M-D Transitional Probabilities to 1-D (Contd)

How about using discrete arrays for continuous random variables?

We can sample using:

$$A_d[i, j] = A_a(i\delta x, j\delta y)$$

where δx, δy can be determined based on the level of accuracy that is required.

Thus, after the algorithm terminates, the accuracy will be at-most δx in the first direction and δy in the second direction.

Defining M-D Probability functions over an M-D Grid

The basic idea is to put larger likelihoods over the points that we would like to put more emphasis on.

For the Hastings-Metropolis algorithm, we will not even need to normalize our probability function. In this case, put larger positive numbers over the regions that we want to sample more densely. To normalize, we simply add up all the values and divide by the total sum so that all the probabilities add up to 1.

If we have a collection of points where we expect the solution to be, we can use distance functions from these points to set up the probability functions.

Defining M-D Probability functions over an M-D Grid

The basic idea is to put larger likelihoods over the points that we would like to put more emphasis on.

For the Hastings-Metropolis algorithm, we will not even need to normalize our probability function. In this case, put larger positive numbers over the regions that we want to sample more densely. To normalize, we simply add up all the values and divide by the total sum so that all the probabilities add up to 1.

If we have a collection of points where we expect the solution to be, we can use distance functions from these points to set up the probability functions.

Defining M-D Probability functions over an M-D Grid

The basic idea is to put larger likelihoods over the points that we would like to put more emphasis on.

For the Hastings-Metropolis algorithm, we will not even need to normalize our probability function. In this case, put larger positive numbers over the regions that we want to sample more densely. To normalize, we simply add up all the values and divide by the total sum so that all the probabilities add up to 1.

If we have a collection of points where we expect the solution to be, we can use distance functions from these points to set up the probability functions.

Defining M-D Probability functions over an M-D Grid

The basic idea is to put larger likelihoods over the points that we would like to put more emphasis on.

For the Hastings-Metropolis algorithm, we will not even need to normalize our probability function. In this case, put larger positive numbers over the regions that we want to sample more densely. To normalize, we simply add up all the values and divide by the total sum so that all the probabilities add up to 1.

If we have a collection of points where we expect the solution to be, we can use distance functions from these points to set up the probability functions.

Defining M-D Probability functions over an M-D Grid (Contd)

In 2-D, we can try $\exp[-a(i-i_0)^2 - b(j-j_0)^2]$ where we can use a, $b > 0$ to define "attraction ranges" around each point (i_0, j_0). For high confidence in (i_0, j_0), we can use large values for a, b.

Note that if we have a large collection of points, we can use clustering to define the distance functions to the centroids.

If we simply put Gaussian around each point, we can show that we can approximate any distribution as the number of points goes to infinity (see any "Pattern Recognition" book by Theodoridis et al, on the use of Parzen Windows in M-D, theorem was actually proven by Prof. Cacoullos in 1962).

Defining M-D Probability functions over an M-D Grid (Contd)

In 2-D, we can try $\exp[-a(i-i_0)^2 - b(j-j_0)^2]$ where we can use a, $b > 0$ to define "attraction ranges" around each point (i_0, j_0). For high confidence in (i_0, j_0), we can use large values for a, b.

Note that if we have a large collection of points, we can use clustering to define the distance functions to the centroids.

If we simply put Gaussian around each point, we can show that we can approximate any distribution as the number of points goes to infinity (see any "Pattern Recognition" book by Theodoridis et al, on the use of Parzen Windows in M-D, theorem was actually proven by Prof. Cacoullos in 1962).

Defining M-D Probability functions over an M-D Grid (Contd)

In 2-D, we can try $\exp[-a(i-i_0)^2 - b(j-j_0)^2]$ where we can use a, $b > 0$ to define "attraction ranges" around each point (i_0, j_0). For high confidence in (i_0, j_0), we can use large values for a, b.

Note that if we have a large collection of points, we can use clustering to define the distance functions to the centroids.

If we simply put Gaussian around each point, we can show that we can approximate any distribution as the number of points goes to infinity (see any "Pattern Recognition" book by Theodoridis et al, on the use of Parzen Windows in M-D, theorem was actually proven by Prof. Cacoullos in 1962).

Outline

Markov Chains

Consider a collection of random variables: $X_0, X_1, \ldots$.

Here, X_n represents the state of a system at time n.

Consider a finite number of states: $1, \ldots, N$.

For transitions from state i to state j, let P_{ij} denote the probability of transition. It is thus assumed to only be a function of the current state.

We have that $\{X_n,\ n \geq 0\}$ constitutes a Markov Chain with transition probabilities P_{ij}.

Markov Chains

Consider a collection of random variables: $X_0, X_1, \ldots$.
Here, X_n represents the state of a system at time n.

Consider a finite number of states: $1, \ldots, N$.

For transitions from state i to state j, let P_{ij} denote the probability of transition. It is thus assumed to only be a function of the current state.

We have that $\{X_n,\ n \geq 0\}$ constitutes a Markov Chain with transition probabilities P_{ij}.

Markov Chains

Consider a collection of random variables: $X_0, X_1, \ldots$.
Here, X_n represents the state of a system at time n.
Consider a finite number of states: $1, \ldots, N$.

For transitions from state i to state j, let P_{ij} denote the probability of transition. It is thus assumed to only be a function of the current state.

We have that $\{X_n,\ n \geq 0\}$ constitutes a Markov Chain with transition probabilities P_{ij}.

Markov Chains

Consider a collection of random variables: $X_0, X_1, \ldots$.
Here, X_n represents the state of a system at time n.
Consider a finite number of states: $1, \ldots, N$.

For transitions from state i to state j, let P_{ij} denote the probability of transition. It is thus assumed to only be a function of the current state.

We have that $\{X_n,\ n \geq 0\}$ constitutes a Markov Chain with transition probabilities P_{ij}.

Markov Chains

Consider a collection of random variables: $X_0, X_1, \ldots$.
Here, X_n represents the state of a system at time n.
Consider a finite number of states: $1, \ldots, N$.

For transitions from state i to state j, let P_{ij} denote the probability of transition. It is thus assumed to only be a function of the current state.

We have that $\{X_n,\ n \geq 0\}$ constitutes a Markov Chain with transition probabilities P_{ij}.

Stationary Probabilities

A Markov Chain is said to be irreducible if we can reach every state from every other state.

For an irreducible Markov chain, let π_j denote the stationary probability that represents the long-run proportion of time that we spent at state j.

They can be found from: $\pi_j = \sum_{i=1}^{N} \pi_i P_{ij}, \quad j = 1, \ldots, N.$

Stationary Probabilities

A Markov Chain is said to be irreducible if we can reach every state from every other state.

For an irreducible Markov chain, let π_j denote the stationary probability that represents the long-run proportion of time that we spent at state j.

They can be found from: $\pi_j = \sum_{i=1}^{N} \pi_i P_{ij}, \quad j = 1, \ldots, N.$

Stationary Probabilities

A Markov Chain is said to be irreducible if we can reach every state from every other state.

For an irreducible Markov chain, let π_j denote the stationary probability that represents the long-run proportion of time that we spent at state j.

They can be found from: $\pi_j = \sum_{i=1}^{N} \pi_i P_{ij}, \quad j = 1, \ldots, N.$

Hastings-Metropolis Algorithm (I of II)

Suppose that we want to simulate a random variable with probability mass function given by:

$$\pi(j) = b(j)/B, \quad j = 1, \ldots, m.$$

Where:

$$B = \sum_{j=1}^{m} b(j), \quad b(j) > 0.$$

To accomplish this, we need to define an irreducible Markov transition probability matrix over $j = 1, \ldots, m$. Here, $P\{X = j\} = q(X_n, j)$ gives the probability that we transition from state X_n to state j.

Hastings-Metropolis Algorithm (I of II)

Suppose that we want to simulate a random variable with probability mass function given by:

$$\pi(j) = b(j)/B, \quad j = 1, \ldots, m.$$

Where:

$$B = \sum_{j=1}^{m} b(j), \quad b(j) > 0.$$

To accomplish this, we need to define an irreducible Markov transition probability matrix over $j = 1, \ldots, m$. Here, $P\{X = j\} = q(X_n, j)$ gives the probability that we transition from state X_n to state j.

Hastings-Metropolis Algorithm (I of II)

Suppose that we want to simulate a random variable with probability mass function given by:

$$\pi(j) = b(j)/B, \quad j = 1, \ldots, m.$$

Where:

$$B = \sum_{j=1}^{m} b(j), \quad b(j) > 0.$$

To accomplish this, we need to define an irreducible Markov transition probability matrix over $j = 1, \ldots, m$. Here, $P\{X = j\} = q(X_n, j)$ gives the probability that we transition from state X_n to state j.

Choosing the transition probability ...

To define $q(X_n, j)$, we can simply define a neighborhood system as before. Then, we can simply go to each neighbor with equal probability.

We do need to make sure that we can reach all the states!

For constrained optimization problems, we need to start with a feasible point, make sure that the neighbors also satisfy the constraints, and we can reach all possible points from the starting point.

This can be done easily by simply looking at all the neighbors, counting the ones that satisfy the constraints, and picking one of them (with equal probability).

Choosing the transition probability ...

To define $q(X_n, j)$, we can simply define a neighborhood system as before. Then, we can simply go to each neighbor with equal probability.

We do need to make sure that we can reach all the states!

For constrained optimization problems, we need to start with a feasible point, make sure that the neighbors also satisfy the constraints, and we can reach all possible points from the starting point.

This can be done easily by simply looking at all the neighbors, counting the ones that satisfy the constraints, and picking one of them (with equal probability).

Choosing the transition probability ...

To define $q(X_n, j)$, we can simply define a neighborhood system as before. Then, we can simply go to each neighbor with equal probability.

We do need to make sure that we can reach all the states!

For constrained optimization problems, we need to start with a feasible point, make sure that the neighbors also satisfy the constraints, and we can reach all possible points from the starting point.

This can be done easily by simply looking at all the neighbors, counting the ones that satisfy the constraints, and picking one of them (with equal probability).

Choosing the transition probability ...

To define $q(X_n, j)$, we can simply define a neighborhood system as before. Then, we can simply go to each neighbor with equal probability.

We do need to make sure that we can reach all the states!

For constrained optimization problems, we need to start with a feasible point, make sure that the neighbors also satisfy the constraints, and we can reach all possible points from the starting point.

This can be done easily by simply looking at all the neighbors, counting the ones that satisfy the constraints, and picking one of them (with equal probability).

Hastings-Metropolis Algorithm (II of II)

Start at state k: $X_0 = k$.

```
While (1) {
  Generate X from P{X = j} = q(X_n, j).
  Generate a uniformly distributed [0,1) random variable U.
  If  U < b(X)q(X, X_n)/[b(X_n)q(X_n, X)]
    Then  X_{n+1} = X
    Else  X_{n+1} = X_n.
  }
```

Make sure to throw away the first set of samples.

Hastings-Metropolis Algorithm (II of II)

Start at state k: $X_0 = k$.

`While` (1) {

`Generate` X from $P\{X = j\} = q(X_n, j)$.

`Generate` a uniformly distributed $[0, 1)$ random variable U.

`If` $U < b(X)q(X, X_n)/[b(X_n)q(X_n, X)]$

`Then` $X_{n+1} = X$

`Else` $X_{n+1} = X_n$.

}

Make sure to throw away the first set of samples.

Hastings-Metropolis Algorithm (II of II)

Start at state k: $X_0 = k$.

`While` (1) {

`Generate` X from $P\{X = j\} = q(X_n, j)$.

`Generate` a uniformly distributed $[0, 1)$ random variable U.

`If` $U < b(X)q(X, X_n)/[b(X_n)q(X_n, X)]$

`Then` $X_{n+1} = X$

`Else` $X_{n+1} = X_n$.

}

Make sure to throw away the first set of samples.

Hastings-Metropolis Algorithm (II of II)

Start at state k: $X_0 = k$.

`While` (1) {

`Generate` X from $P\{X = j\} = q(X_n, j)$.

`Generate` a uniformly distributed $[0, 1)$ random variable U.

`If` $U < b(X)q(X, X_n)/[b(X_n)q(X_n, X)]$

`Then` $X_{n+1} = X$

`Else` $X_{n+1} = X_n$.

}

Make sure to throw away the first set of samples.

Hastings-Metropolis Algorithm (II of II)

Start at state k: $X_0 = k$.

`While` (1) {

`Generate` X from $P\{X = j\} = q(X_n, j)$.

`Generate` a uniformly distributed $[0, 1)$ random variable U.

`If` $U < b(X)q(X, X_n)/[b(X_n)q(X_n, X)]$

`Then` $X_{n+1} = X$

`Else` $X_{n+1} = X_n$.

}

Make sure to throw away the first set of samples.

The Gibbs Sampler (I of III)

Suppose that we want to generate a multivariate random variable $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with probability density function $p(\mathbf{x})$, known up to a multiplicative constant.

For example, we know (or specify) $g(\mathbf{x})$ and we do know that $p(\mathbf{x}) = Cg(\mathbf{x})$, for some unknown $C > 0$.

So $g(\mathbf{x})$ can simply be a positive function that we specify!

We can do this using a Markov chain, where the transition probabilities are the one-dimensional (!) conditional probabilities $P\{X = x\}$ given by:

$$P\{X_i = x \,|\, X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n\}.$$

The Gibbs Sampler (I of III)

Suppose that we want to generate a multivariate random variable $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with probability density function $p(\mathbf{x})$, known up to a multiplicative constant.

For example, we know (or specify) $g(\mathbf{x})$ and we do know that $p(\mathbf{x}) = Cg(\mathbf{x})$, for some unknown $C > 0$.

So $g(\mathbf{x})$ can simply be a positive function that we specify!

We can do this using a Markov chain, where the transition probabilities are the one-dimensional (!) conditional probabilities $P\{X = x\}$ given by:

$$P\{X_i = x \,|\, X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n\}.$$

The Gibbs Sampler (I of III)

Suppose that we want to generate a multivariate random variable $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with probability density function $p(\mathbf{x})$, known up to a multiplicative constant.

For example, we know (or specify) $g(\mathbf{x})$ and we do know that $p(\mathbf{x}) = Cg(\mathbf{x})$, for some unknown $C > 0$.

So $g(\mathbf{x})$ can simply be a positive function that we specify!

We can do this using a Markov chain, where the transition probabilities are the one-dimensional (!) conditional probabilities $P\{X = x\}$ given by:

$$P\{X_i = x \,|\, X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n\}.$$

The Gibbs Sampler (I of III)

Suppose that we want to generate a multivariate random variable $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with probability density function $p(\mathbf{x})$, known up to a multiplicative constant.

For example, we know (or specify) $g(\mathbf{x})$ and we do know that $p(\mathbf{x}) = Cg(\mathbf{x})$, for some unknown $C > 0$.

So $g(\mathbf{x})$ can simply be a positive function that we specify!

We can do this using a Markov chain, where the transition probabilities are the one-dimensional (!) conditional probabilities $P\{X = x\}$ given by:

$$P\{X_i = x \,|\, X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n\}.$$

The Gibbs Sampler (II of III)

The key is to define the Markov chain transition probabilities to move to each conditional coordinate with equal probability:

$$q(\mathbf{x}, \mathbf{y}) = \frac{1}{n} P\{X_i = x \,|\, X_j = x_j, j \neq i\}$$
$$= \frac{p(\mathbf{y})}{nP\{X_j = x_j, j \neq i\}}$$

where

n denotes the number of variables.

The Gibbs Sampler (II of III)

The key is to define the Markov chain transition probabilities to move to each conditional coordinate with equal probability:

$$q(\mathbf{x}, \mathbf{y}) = \frac{1}{n} P\{X_i = x \,|\, X_j = x_j, j \neq i\}$$
$$= \frac{p(\mathbf{y})}{nP\{X_j = x_j, j \neq i\}}$$

where

n denotes the number of variables.

The Gibbs Sampler (II of III)

The key is to define the Markov chain transition probabilities to move to each conditional coordinate with equal probability:

$$q(\mathbf{x}, \mathbf{y}) = \frac{1}{n} P\{X_i = x \,|\, X_j = x_j, j \neq i\}$$
$$= \frac{p(\mathbf{y})}{nP\{X_j = x_j, j \neq i\}}$$

where

n denotes the number of variables.

The Gibbs Sampler (III of III)

Based on the transition probabilities, we use the Hastings-Metropolis algorithm and accept the generated variable:

Start at state k: $X_0 = k$.

For $k = 1, 2, \ldots$ {

Generate i from a uniform distribution of $1, 2, \ldots, n$.

Let $Y = X_k$ be the previous N-dim sample.

Generate X from the 1-dim $P\{X_i = x \,|\, X_j = y_j, j \neq i\}$

Set $X_{k+1} = X$

}

Make sure to throw away a large number of the initial set of samples.

The Gibbs Sampler (III of III)

Based on the transition probabilities, we use the Hastings-Metropolis algorithm and accept the generated variable:

Start at state k: $X_0 = k$.

For $k = 1, 2, \ldots$ {

Generate i from a uniform distribution of $1, 2, \ldots, n$.

Let $Y = X_k$ be the previous N-dim sample.

Generate X from the 1-dim $P\{X_i = x \,|\, X_j = y_j, j \neq i\}$

Set $X_{k+1} = X$

}

Make sure to throw away a large number of the initial set of samples.

The Gibbs Sampler (III of III)

Based on the transition probabilities, we use the Hastings-Metropolis algorithm and accept the generated variable:

Start at state k: $X_0 = k$.

For $k = 1, 2, \ldots$ {

Generate i from a uniform distribution of $1, 2, \ldots, n$.

Let $Y = X_k$ be the previous N-dim sample.

Generate X from the 1-dim $P\{X_i = x \,|\, X_j = y_j, j \neq i\}$

Set $X_{k+1} = X$

}

Make sure to throw away a large number of the initial set of samples.

The Gibbs Sampler (III of III)

Based on the transition probabilities, we use the Hastings-Metropolis algorithm and accept the generated variable:

Start at state k: $X_0 = k$.
For $k = 1, 2, \ldots$ {
  Generate i from a uniform distribution of $1, 2, \ldots, n$.
  Let $Y = X_k$ be the previous N-dim sample.
  Generate X from the 1-dim $P\{X_i = x \,|\, X_j = y_j, j \neq i\}$
  Set $X_{k+1} = X$
}

Make sure to throw away a large number of the initial set of samples.

The Gibbs Sampler (III of III)

Based on the transition probabilities, we use the Hastings-Metropolis algorithm and accept the generated variable:

Start at state k: $X_0 = k$.

For $k = 1, 2, \ldots$ {

Generate i from a uniform distribution of $1, 2, \ldots, n$.

Let $Y = X_k$ be the previous N-dim sample.

Generate X from the 1-dim $P\{X_i = x \,|\, X_j = y_j, j \neq i\}$

Set $X_{k+1} = X$

}

Make sure to throw away a large number of the initial set of samples.

The Gibbs Sampler (III of III)

Based on the transition probabilities, we use the Hastings-Metropolis algorithm and accept the generated variable:

Start at state k: $X_0 = k$.

For $k = 1, 2, \ldots$ {

Generate i from a uniform distribution of $1, 2, \ldots, n$.

Let $Y = X_k$ be the previous N-dim sample.

Generate X from the 1-dim $P\{X_i = x \,|\, X_j = y_j, j \neq i\}$

Set $X_{k+1} = X$

}

Make sure to throw away a large number of the initial set of samples.

The Gibbs Sampler (III of III)

Based on the transition probabilities, we use the Hastings-Metropolis algorithm and accept the generated variable:

Start at state k: $X_0 = k$.
For $k = 1, 2, \ldots$ {
  Generate i from a uniform distribution of $1, 2, \ldots, n$.
  Let $Y = X_k$ be the previous N-dim sample.
  Generate X from the 1-dim $P\{X_i = x \,|\, X_j = y_j, j \neq i\}$
  Set $X_{k+1} = X$
  }

Make sure to throw away a large number of the initial set of samples.