Introduction of Expectation-Maximization Algorithm, Cross-Entropy Method and Genetic Algorithm
Outline

- Introduction of Expectation-Maximization Algorithm
- Introduction of Genetic Algorithm
- Introduction of Cross-Entropy Method
Introduction of Expectation-Maximum Algorithm

Wireless Information Transmission System Lab.
Institute of Communications Engineering
National Sun Yat-sen University

2010/08/03
顏愷緯
Expectation-Maximization Algorithm

- Expectation-maximization (EM) algorithm is found and given its name in 1970s.
- EM algorithm is an algorithm for finding maximum likelihood (ML) estimates of parameters in probabilistic models, where the model depends on unobserved latent variables.
- It is an efficient and quite useful algorithm for finding the ML estimation of parameters when the data model is easy to present.
EM alternates between performing the following two steps:

- expectation (E) step, which computes an expectation of the likelihood by including the latent variables as if they were observed.
- maximization (M) step, which computes the maximum likelihood estimates of the parameters by maximizing the expected likelihood found on the E step.

The parameters found on the M step are then used to begin another E step, and the process is repeated.
Assume that $X$ is the observable random vector, $z$ is the missing random vector.

In maximum likelihood (ML) estimation for $\theta$, we want to find $\theta$ such that $p(X|\theta)$ is a maximum. The log likelihood function is known as

$$L(\theta) = \ln p(X|\theta).$$

The EM algorithm is an iterative process for maximizing $L(\theta)$. 
We would like to find a new $\theta$ such that

$$L(\theta) > L(\theta^n),$$

where $\theta^n$ means the value of the $n$th iteration.

The probability $p(X|\theta)$ can be written in terms of the hidden variables $z$ as

$$p(X|\theta) = \sum_z p(X|z, \theta)p(z|\theta).$$

Using Jensen’s inequality, it was shown that,

$$\ln \sum_{i=1}^n \lambda_i v_i \geq \sum_{i=1}^n \lambda_i \ln(v_i).$$
Then the equation $L(\theta) - L(\theta^n)$ can be expressed as

$$L(\theta) - L(\theta^n) = \ln \left( \sum_z p(X|z, \theta)p(z|\theta) \right) - \ln p(X|\theta^n)$$

$$= \ln \left( \sum_z p(X|z, \theta)p(z|\theta) \frac{p(z|X, \theta^n)}{p(z|X, \theta^n)} \right) - \ln p(X|\theta^n)$$

$$= \ln \left( \sum_z p(z|X, \theta^n) \frac{p(X|z, \theta)p(z|\theta)}{p(z|X, \theta^n)} \right) - \ln p(X|\theta^n)$$

$$\geq \sum_z p(z|X, \theta^n) \ln \left( \frac{p(X|z, \theta)p(z|\theta)}{p(z|X, \theta^n)} \right) - \ln p(X|\theta^n)$$

$$= \sum_z p(z|X, \theta^n) \ln \left( \frac{p(X|z, \theta)p(z|\theta)}{p(z|X, \theta^n)} \right) - \left[ \sum_z p(z|X, \theta^n) \right] \ln p(X|\theta^n)$$

$$= \sum_z p(z|X, \theta^n) \ln \left( \frac{p(X|z, \theta)p(z|\theta)}{p(z|X, \theta^n)p(X|\theta^n)} \right).$$
For convenience, we define the following equation

\[ \Delta(\theta|\theta^n) \equiv L(\theta^n) + \sum_z p(z|X,\theta^n) \ln \left( \frac{p(X|z,\theta)p(z|\theta)}{p(z|X,\theta^n)p(X|\theta^n)} \right) \leq L(\theta). \]

Since \( L(\theta) \geq \Delta(\theta|\theta^n) \), it can be said that \( \Delta(\theta|\theta^n) \) is bounded by the likelihood function \( L(\theta) \) and that the value of the functions \( L(\theta) \) and \( \Delta(\theta|\theta^n) \) are equal at current estimate because that

\[ \Delta(\theta^n|\theta^n) = L(\theta^n) + \sum_z p(z|X,\theta^n) \ln \left( \frac{p(X|z,\theta^n)p(z|\theta^n)}{p(z|X,\theta^n)p(X|\theta^n)} \right) = L(\theta^n). \]
Therefore any $\theta$ which increases $\Delta(\theta|\theta^n)$ also increase the $L(\theta)$. In order to achieve the greatest possible increase in the value of $L(\theta)$, the EM algorithm calls for selecting $\theta$ such that $\Delta(\theta|\theta^n)$ is maximized. Formally, we have

$$
\theta^{n+1} = \arg \max_{\theta} \left\{ \Delta(\theta|\theta^n) \right\}
$$

$$
= \arg \max_{\theta} \left\{ L(\theta^n) + \sum_z p(z|X,\theta^n) \ln \left( \frac{p(X|z,\theta) p(z|\theta)}{p(z|X,\theta^n) p(X|\theta^n)} \right) \right\}.
$$
Drop the term $L(\theta^n)$ which is constant with respect to $\theta$. Then we can obtain

$$\theta^{n+1} = \arg \max_{\theta} \left\{ \sum_z p(z|X,\theta^n) \ln p(X|z,\theta) p(z|\theta) \right\}$$

$$= \arg \max_{\theta} \left\{ \sum_z p(z|X,\theta^n) \ln p(X,z|\theta) \right\}$$

$$= \arg \max_{\theta} \left\{ \mathbb{E}_{z|X,\theta^n} \left[ \ln p(X,z|\theta) \right] \right\}.$$
Thus, the EM algorithm consists of iterating the:

- **E-step**: Determine the conditional expectation

\[ Q(\theta^{n+1} | \theta^n) = E_{z|X,\theta^n} \left[ \ln p(X, z | \theta) \right] \]

- **M-step**: Maximize this expression with respect to \( \theta \).

We can consider the set of observations \( X \) as being “incomplete” in relation to the “complete” set \( \{X, z\} \).
If the likelihood function is bounded, and if $Q(\theta^{n+1} | \theta^n)$ is continuous, the log-likelihood function $\ln p(X | \theta)$ converges to a stationary point (a local maximum).

When the likelihood function has several maxima, the EM algorithm is not guaranteed to converge to the global maximum.

EM is particularly useful when maximum likelihood estimation of a complete data model is easy. If Closed form estimators exist, the M step is often trivial.
Expectation-Maximization Algorithm
We transmit the complex symbol $s_t$ and we receive $y_{tm}$ on receive antennas. The received signal can be expressed as

$$y_{tm} = h_m s_t, \quad t = 1, 2, \ldots, T; \quad m = 1, 2, \ldots, M.$$ 

where $h_m$ is the complex-valued fading coefficient between the transmitter and the $m$th receiver.

The fading coefficients are assumed to be independent and $CN(0, 1)$ distributed.

Let $T$ be the length during which channel state information (CSI) remains constant.
If one block of $T$ symbols are sent, the received signal blocks by $M$ receive antennas can be expressed by a matrix $Y$ of $M \times T$. It can be written as

$$Y = hs + N$$

The noise is assumed to be white, i.e., $E \{ N_{t_1} N_{t_2}^H \} = \Sigma \delta(t_1 - t_2)$, where $\Sigma \in \mathbb{C}^{M \times M}$ is the noise covariance matrix.
We define a complete data, \( X = (Y, h) \), for the parameter that we want to estimate.

- **E-step:** The E-step of EM algorithm requires the calculation of \( Q(s | s^{[k]}) \) where

\[
Q(s | s^{[k]}) = E\{\ln f(X | s) | Y, s^{[k]}\}.
\]

- **M-step:** M-step is to find the \( s \) that maximize the \( Q \) function as

\[
s^{[k+1]} = \arg\max_s Q(s | s^{[k]}).
\]
Introduction of Cross-Entropy Method
The Cross-Entropy Method was originally developed as a simulation method for the estimation of rare event probabilities:

\[
\text{Estimate } P(S(X) \geq \gamma)
\]

- \( X \): random vector/process taking values in some set \( \chi \).
- \( S \): real-valued function on \( \chi \).

It was soon realized that the CE Method could also be used as an optimization method:

\[
\text{Determine } \max_{x \in \chi} S(X)
\]
Cross-Entropy Method

Many real-world optimization problems can be formulated mathematically as either (or both)

- Locate some element $x^*$ in a set $\chi$ such that
  \[ S(x^*) \geq S(x) \text{ for all } x \in \chi \]
  where $S$ is an objective function or performance measure defined on $\chi$.
- Find $\gamma^* = S(x^*)$, the globally maximal value of the function.

The CE method involves an iterative procedure where each iteration can be broken into two phase:

- Generate a random, data sample according to specified mechanism.
- Update the parameters of the random mechanism based on the data to produce a “better” sample in the next iteration.
Cross-Entropy Method

General CE Algorithm:

1) Initialize the parameters

2) Generate the random sample from the density function which is updated in previous iteration and calculate the fitness value.

3) Update the probability density function of the random sample based on the acquired information to produce a better solution in the next iteration.

4) Return to Step 2 until some stopping criterion is met.
Cross-Entropy Method (Example 1)

◊ Consider a binary vector \( y = \left[ y_1, y_2, \ldots, y_n \right] \). Suppose that we don’t know which components of \( y \) are 0 and which are 1.

◊ We have a oracle which for each binary input vector \( x = \left[ x_1, x_2, \ldots, x_n \right] \) returns the performance or response,

\[
S(x) = n - \sum_{j=1}^{n} |x_j - y_j|,
\]

representing the number of matches between the elements of \( x \) and \( y \).
Cross-Entropy Method (Example 1)

\[ S(x) = n - \sum_{j=1}^{n} |x_j - y_j| \]
Cross-Entropy Method (Example 1)

1) Start with iteration number $t = 1$ and probability $p_0 = [0.5, \ldots, 0.5]$.  

2) Draw a sample $X_1, \ldots, X_N$ of Bernoulli vectors with success probability vector $p_{t-1}$.  

3) Calculate the performances $S(X_i)$ for all $i$, and order them from the smallest to biggest. Let $\gamma_t$ be $(1-\rho)$ sample quantile of the performances: $\gamma_t = S\left(\lfloor(1-\rho)N\rfloor\right)$. 

4) Use the same sample to calculate \( p_t = \left[ p_{t,1}, p_{t,2}, \cdots, p_{t,n} \right] \) via

\[
p_{t,j} = \frac{\sum_{j=1}^{N} I_{\{s(x_i) \geq \gamma_t\}} I_{\{x_{ij} = 1\}}}{\sum_{j=1}^{N} I_{\{s(x_i) \geq \gamma_t\}}},
\]

\( j = 1, \ldots, n \), where \( X_i = [X_{i1}, X_{i2}, \cdots X_{in}] \), and increase \( t \) by 1.

5) If the stopping criterion is met, then stop; otherwise set \( t = t + 1 \) and reiterate from step 2 and step 3.
Cross-Entropy Method (Example 1)

As an example consider the case \( n = 10, N = 50 \) and \( \rho = 0.1 \) where \( y = [1,1,1,1,0,0,0,0,0,0] \).

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \gamma_t )</th>
<th>( p_t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>0.60</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>0.80</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Cross-Entropy Method (Example 2)

\[ S(x) = e^{-(x-2)^2} + 0.8 e^{- (x+2)^2} \]
Cross-Entropy Method (Example 2)

S = inline(‘exp(-(x - 2).^2)) + 0.8 * exp(-(x + 2).^2)’);
mu = -10; sigma = 10; rho = 0.1; N = 100; eps = 1E-3; t = 0;

while sigma > eps
    t = t + 1;
    x = mu + sigma * randn(N,1);
    SX = S(x); % compute the performance
    mu = mean(sortSX((1 - rho) * N:N,1));
    sigma = std(sortSX((1 - rho) * N:N,1));
end
Cross-Entropy Method (Example 2)
Cross-Entropy Method

◊ Some strategies to improve the performance of CE method:

◊ Elite reservation.
◊ Smoothing factor.
◊ The number of $\rho$. 
Introduction of Genetic Algorithm
Genetic Algorithm

Genetic Algorithm (GA), first introduced by John Holland in the early seventies, is the powerful stochastic algorithm based on the principles of natural selection and natural genetics.

They are used as optimization methods and have shown their effectiveness in various problems.

GAs evolves solutions in an iterative manner by applying genetic operators to a pool of candidate solutions.

To solve a problem, a GA maintains a population of individuals and probabilistically modifies the population by some genetic operators such as selection, crossover and mutation, with the intent of seeking a suboptimal solution to the problem.
Genetic Algorithm

Start

Initial population

Compute the fitness function

Genetic Operators (selection)

Genetic Operators (mutation and crossover)

Terminal criteria

No

Yes

end

2
Genetic Algorithm

◊ In GA, each individual in a population is usually coded as a fixed-length binary string. The length of the string depends on the domain of the parameters and the required precision.

◊ For example, if the domain of the parameter x is [-2,5] and the precision requirement is six places after the decimal point, then the domain [-2,5] should be divided into 7,000,000 equal size ranges.

◊ That implies the length of the string requires to be 23, for the reason that

\[ 4194304 = 2^{22} < 7000000 < 2^{23} = 8388608. \]
Genetic Algorithm

- The initial process is quite simple. We create a population of individuals, where individual in a population is a binary string with a fixed-length, and every bit of the binary string is initialized randomly.

- In each generation for which the GA is run, each individual in the population is evaluated against the unknown environment. The fitness values are associated with the values of objective function.
Genetic Algorithm

- To perform genetic operators, one must select individuals in the population to be operated on. The selection strategy is chiefly based on the fitness level of the individuals actually presented in the population.

- There are many different selection strategies based on fitness, such as Roulette Wheel Selection, Elitism Strategy and Tournament Selection.
Genetic Algorithm

◊ Roulette Wheel Selection:

Step1. evaluate the fitness values for every individual, \( s_k, k = 1, 2, \ldots, N \).

Step2. evaluate the total fitness values of all the individuals, \( T \).

Step3. calculate ratio of fitness of each individual.

| \( x_1 \) | 25 | 0.1116 | 1 | \( x_1 \) |
| \( x_2 \) | 40 | 0.1786 | 2 | \( x_2 \) |
| \( x_3 \) | 30 | 0.1339 | 1 | \( x_2 \) |
| \( x_4 \) | 33 | 0.1473 | 1 | \( x_3 \) |
| \( x_5 \) | 28 | 0.1250 | 1 | \( x_4 \) |
| \( x_6 \) | 15 | 0.0669 | 1 | \( x_5 \) |
| \( x_7 \) | 50 | 0.2232 | 2 | \( x_6 \) |
| \( x_8 \) | 2  | 0.0089 | 0 | \( x_7 \) |
| \( x_9 \) | 1  | 0.0044 | 0 | \( x_7 \) |
Genetic Algorithm

- The crossover operator starts with two selected individuals and then the crossover point is selected randomly according to a given probability of crossover.

- The second genetic operator, mutation, introduces random changes in structures in the population, and it may occasionally have beneficial results: escaping from a local optimum.

- In binary form GA, mutation is just to negate every bit of the strings, i.e., changes a 1 to 0 and vice versa, with given probability of mutation.
Genetic Algorithm

One-Point Crossover

0101110  10100110
1001101  10011010

Two-Point Crossover

0101110  10100110  1011011
1110111  00001111  0111011

0101110  10100110
1001101  10011010
<table>
<thead>
<tr>
<th>Binary form</th>
<th>Real-number form</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110111 0001111 0111011</td>
<td>119 15 59</td>
</tr>
<tr>
<td>1110111 0000111 0111011</td>
<td>119 7 59</td>
</tr>
</tbody>
</table>
[x,fval,exitflag,output,population,scores] =
   ga(@fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon,options)

◊ It finds a local minimum \( x \) to \( \text{fitnessfcn} \), subject to the linear inequalities \( A \times x \leq b \) as well as the linear equalities \( Aeq \times x = beq \). \( \text{fitnessfcn} \) accepts input \( x \) and returns a scalar function value evaluated at \( x \).

◊ If the problem has \( m \) linear inequality constraints and \( n \) variables, then
  ◦ \( A \) is a matrix of size \( m \)-by-\( n \) variables.
  ◦ \( b \) is a vector of length \( m \).
Global Optimization Toolbox

◊ UB and LB define a set of lower and upper bounds on the design variables, x, so that a solution is found in the range $LB \leq x \leq UB$. Use empty matrices for LB and UB if no bounds exist.

◊ options = gaoptimset('param1',value1,'param2',value2,...) creates a structure called options and sets the value of param1 to value1, param2 to value2, and so on.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitnessfcn</td>
<td>Fitness function</td>
</tr>
<tr>
<td>nvars</td>
<td>Number of design variables</td>
</tr>
<tr>
<td>Aineq</td>
<td>A matrix for linear inequality constraints</td>
</tr>
<tr>
<td>Bineq</td>
<td>b vector for linear inequality constraints</td>
</tr>
<tr>
<td>Aeq</td>
<td>A matrix for linear equality constraints</td>
</tr>
<tr>
<td>Beq</td>
<td>b vector for linear equality constraints</td>
</tr>
<tr>
<td>lb</td>
<td>Lower bound on x</td>
</tr>
<tr>
<td>ub</td>
<td>Upper bound on x</td>
</tr>
<tr>
<td>nonlcon</td>
<td>Nonlinear constraint function</td>
</tr>
<tr>
<td>options</td>
<td>Options structure created using gaoptimset or the Optimization Tool</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreationFcn</td>
<td>Handle to the function that creates the initial population</td>
</tr>
<tr>
<td>CrossoverFraction</td>
<td>The fraction of the population at the next generation, not including elite children, that is created by the crossover function</td>
</tr>
<tr>
<td>MutationFcn</td>
<td>Handle to the function that produces mutation children</td>
</tr>
<tr>
<td>EliteCount</td>
<td>Positive integer specifying how many individuals in the current generation are guaranteed to survive to the next generation. Not used in gamultiobj.</td>
</tr>
<tr>
<td>Generations</td>
<td>Positive integer specifying the maximum number of iterations before the algorithm halts</td>
</tr>
<tr>
<td>InitialPenalty</td>
<td>Initial value of penalty parameter</td>
</tr>
<tr>
<td>InitialPopulation</td>
<td>Initial population used to seed the genetic algorithm; can be partial</td>
</tr>
<tr>
<td>PopulationSize</td>
<td>Size of the population</td>
</tr>
<tr>
<td>PopulationType</td>
<td>String describing the data type of the population</td>
</tr>
<tr>
<td>StallGenLimit</td>
<td>Positive integer. The algorithm stops if there is no improvement in the objective function for StallGenLimit consecutive generations.</td>
</tr>
<tr>
<td>TolCon</td>
<td>Positive scalar. TolCon is used to determine the feasibility with respect to nonlinear constraints.</td>
</tr>
<tr>
<td>TolFun</td>
<td>Positive scalar. The algorithm runs until the cumulative change in the fitness function value over StallGenLimit is less than TolFun.</td>
</tr>
</tbody>
</table>
Global Optimization Toolbox (Example 1)

\[ \text{Max } f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2) \]
\[-3.0 \leq x_1 \leq 12.1, \ 4.1 \leq x_2 \leq 5.8 \]
function [ f ] = objectfunc_conti(x)
    f = -(21.5 + x(1)*sin(4*pi*x(1)) + x(2)*sin(20*pi*x(2)));

Lb = [-3 ; 4.1];
Ub = [ 12.1 ; 5.8];
options = gaoptimset(@ga);
options = gaoptimset(options,...
    'MutationFcn',{@mutationadaptfeasible,0.001},... 
    'CrossoverFraction',0.8 ,... 
    'PopInitRange', [-3 4.1; 12.1 5.8] ,... 
    'PopulationSize',100 ,... 
    'Generation',1000,...
    'PlotFcns',{ @gaplotbestf, @gaplotbestindiv });
[x fitval] = ga(@objectfunc_conti,2,[],[],[],[],Lb,Ub,[],options);
Example

\[ y = [1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \]

**Figure 2:** A black box for decoding vector \( y \).

\[ S(x) = n - \sum_{j=1}^{n} |x_j - y_j| \]
function [ f ] = objectunc_dis(x)
    y = [1 1 1 1 1 0 0 0 0 0]
    for ii = 1 :10
        Sx(ii) = abs(x(ii)-y(ii));
    end
    f = sum(Sx);

options = gaoptimset(@ga);
options = gaoptimset(options,...
    'PopulationSize',50,...
    'Generation',3,...
    'PopulationType','bitstring');
[x,fval,exitflag,output,population,scores] =
    ga(@objectunc_dis,10,[],[],[],[],[],[],[],options);
Assignment

Wireless Information Transmission System Lab.
Institute of Communications Engineering
National Sun Yat-sen University

2012/07/23
顏愷緯
DSI Method

![Diagram of DSI Method]

<table>
<thead>
<tr>
<th>$X$</th>
<th>IFFT Input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Divided data $D$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Sequence

| 1 | 2 | ... | $L$ | 1 | ... | $M$ |
DSI Method

- Simulation condition
  - BPSK
  - Total subcarriers = 64
  - Data subcarriers = 56
  - DSI subcarriers = 8
  - Oversample = 4
DSI Method

- GA and CE Method
  - Population size = 10
  - Maximum iteration = 6
  - Elitist = 1
  - Mutation probability = 0.8
  - Crossover = 0.3
  - Fitness function:

\[
PAPR = \max_{0 \leq n \leq \text{LN}-1} \frac{|x[n]|^2}{E[|x|^2]}\]

<table>
<thead>
<tr>
<th>X IFFT Input</th>
<th>Divided data</th>
<th>Dummy bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>L</td>
<td>1 2</td>
</tr>
</tbody>
</table>

\[
PAPR = \max_{0 \leq n \leq \text{LN}-1} \frac{|x[n]|^2}{E[|x|^2]}\]
DSI Method

DSI simulation with GA and CE method

PAPR

CCDF

Original
GA
CE

$PAPR_0$
DSI Method

Reference: