A Matlab Program to Calculate the Maximum Entropy Distributions

Article · July 2000

DOI: 10.1007/978-94-017-2219-3_16 · Source: CiteSeer

1 author:

Ali Mohammad-Djafari
French National Centre for Scientific Research
476 PUBLICATIONS **3,182** CITATIONS
SEE PROFILE

A Matlab Program to Calculate the Maximum Entropy Distributions

Ali Mohammad-Djafari

Laboratoire des Signaux et Systèmes,
Supélec, Plateau de Moulon, 91192 Gif-sur-Yvette, France

INTRODUCTION

Shannon (1948) indicated how maximum entropy (ME) distributions can be derived by a straigtforward application of the calculus of variations technique. He defined the entropy of a probability density function $p(x)$ as

$$H = -\int p(x) \ln p(x) \, \mathrm{d}x \tag{1}$$

Maximizing H subject to various side conditions is well–known in the literature as a method for deriving the forms of minimal information prior distributions; e.g. Jaynes (1968) and Zellner (1977). Jaynes (1982) has extensively analyzed examples in the discrete case, while in Lisman and Van Znylen (1972), Rao (1973) and Gokhale (1975), Kagan, Linjik continuous cases are considered. In the last case, the problem, in its general form, is the following

$$\begin{aligned} \text{maximize} \quad H &= -\int p(x) \ln p(x) \, \mathrm{d}x \\ \text{subject to} \quad \mathrm{E}\{\phi_n(x)\} &= \int \phi_n(x) \, p(x) \, \mathrm{d}x = \mu_n, \quad n = 0, \ldots, N \end{aligned} \tag{2}$$

where $\mu_0 = 1$, $\phi_0(x) = 1$ and $\phi_n(x), n = 0, \ldots, N$ are N known functions, and $\mu_n, n = 0, \ldots, N$ are the given expectation data. The classical solution of this problem is given by

$$p(x) = \exp\left[-\sum_{n=0}^{N} \lambda_n \, \phi_n(x)\right] \tag{3}$$

The $(N+1)$ Lagrangien parameters $\boldsymbol{\lambda} = [\lambda_0, \ldots, \lambda_n]$ are obtained by solving the following set of $(N+1)$ nonlinear equations

$$G_n(\boldsymbol{\lambda}) = \int \phi_n(x) \exp\left[-\sum_{n=0}^{N} \lambda_n \, \phi_n(x)\right] \mathrm{d}x = \mu_n, \quad n = 0, \ldots, N \tag{4}$$

The distributions defined by (3) form a great number of known distributions which are obtained by choosing the appropriate N and $\phi_n(x), n = 0, \ldots, N$. In general $\phi_n(x)$ are either the powers of x or the logarithm of x. See Mukhrejee and Hurst (1984), Zellner (1988), Mohammad–Djafari (1990) for many other examples and discussions. Special cases have been extensively analyzed and used by many authors. When $\phi_n(x) = x^n, n = 0, \ldots, N$ then $\mu_n, n = 0, \ldots, N$ are the given N moments of the distribution. See, for example, Zellner (1988) for a numerical implementation in the case $N = 4$.

In this communication we propose three programs written in MATLAB to solve the system of equations (4). The first is a general program where $\phi_n(x)$ can be any functions. The second is a special case where $\phi_n(x) = x^n, n = 0, \ldots, N$. In this case the μ_n are the geometrical moments of $p(x)$. The third is a special case where $\phi_n(x) = \exp(-jn\omega x), n = 0, \ldots, N$. In this case the μ_n are the trigonometrical moments (Fourier components) of $p(x)$. We give also some examples to illustrate the usefullness of these programs.

PRINCIPLE OF THE METHOD

We have seen that the solution of the standard ME problem is given by (3) in which the Lagrange multipliers $\boldsymbol{\lambda}$ are obtained by solving the nonlinear equations (4). In general, these equations are solved by the standard Newton method which consists of expanding $G_n(\boldsymbol{\lambda})$ in Taylor's series around trial values of the *lambda*'s, drop the quadratic and higher order terms, and solve the resulting linear system iteratively. We give here the details of the numerical method that we implemented. When developing the $G_n(\boldsymbol{\lambda})$ in equations (4) in first order Taylor's series around the trial $\boldsymbol{\lambda}^0$, the resulting linear equations are given by

$$G_n(\boldsymbol{\lambda}) \cong G_n(\boldsymbol{\lambda}^0) + (\boldsymbol{\lambda} - \boldsymbol{\lambda}^0)^t \left[\mathbf{grad}\, G_n(\boldsymbol{\lambda})\right]_{(\boldsymbol{\lambda}=\boldsymbol{\lambda}^0)} = \mu_n, \quad n = 0, \ldots, N \tag{5}$$

Noting the vectors $\boldsymbol{\delta}$ and $\boldsymbol{v}$ by

$$\boldsymbol{\delta} = \boldsymbol{\lambda} - \boldsymbol{\lambda}^0$$

$$\boldsymbol{v} = \left[\mu_0 - G_0(\boldsymbol{\lambda}^0), \ldots, \mu_N - G_N(\boldsymbol{\lambda}^0)\right]^t$$

and the matrix $\boldsymbol{G}$ by

$$\boldsymbol{G} = \left[\begin{array}{c} \\ g_{nk} \\ \\ \end{array}\right] = \left[\begin{array}{c} \\ \frac{\partial G_n(\boldsymbol{\lambda})}{\partial \lambda_k} \\ \\ \end{array}\right]_{(\boldsymbol{\lambda}=\boldsymbol{\lambda}^0)} \qquad n, k = 0, \ldots, N \tag{6}$$

then equations (5) become

$$\boldsymbol{G}\,\boldsymbol{\delta} = \boldsymbol{v} \tag{7}$$

This system is solved for $\boldsymbol{\delta}$ from which we drive $\boldsymbol{\lambda} = \boldsymbol{\lambda}^0 + \boldsymbol{\delta}$, which becomes our new initial vector $\boldsymbol{\lambda}^0$ and the iterations continue until $\boldsymbol{\delta}$ becomes appropriately small. Note that the matrix $\boldsymbol{G}$ is a symmetric one and we have

$$g_{nk} = g_{kn} = -\int \phi_n(x)\,\phi_k(x)\,\exp\left[-\sum_{n=0}^{N} \lambda_n\,\phi_n(x)\right] \mathrm{d}x \quad n,k = 0,\ldots,N \tag{8}$$

So in each iteration we have to calculate the $N(N-1)/2$ integrals in the equation (8). The algorithm of the general Maximum Entropy problem is then as follows:

1. Define the range and the discretization step of x `(xmin, xmax,dx).`
2. Write a function to calculate $\phi_n(x), n = 0,\ldots,N$ `(fin_x).`
3. Start the iterative procedure with an initial estimate $\boldsymbol{\lambda}^0$ `(lambda0).`
4. Calculate the $(N+1)$ integrals in equations (4) and the $N(N-1)/2$ distinct elements g_{nk} of the matrix $\boldsymbol{G}$ by calculating the integrals in the equations(8) `(Gn, gnk).`
5. Solve the equation (7) to find $\boldsymbol{\delta}$ `(delta).`
6. Calculate $\boldsymbol{\lambda} = \boldsymbol{\lambda}^0 + \boldsymbol{\delta}$ and go back to step 3 until $\boldsymbol{\delta}$ becomes negligible.

The calculus of the integrals in equations (4) and (8) can be made by a univariate Simpson's method. We have used a very simplified version of this method.

Case of geometrical moments

Now consider the special case of moments problem where $\phi_n(x) = x^n, \quad n = 0,\ldots,N$. In this case equations (3), (4) and (8) become

$$p(x) = \exp\left[-\sum_{m=0}^{N} \lambda_m\,x^m\right] \tag{9}$$

$$G_n(\boldsymbol{\lambda}) = \int x^n \exp\left[-\sum_{m=0}^{N} \lambda_m\,x^m\right] \mathrm{d}x = \mu_n, \quad n = 0,\ldots,N \tag{10}$$

$$g_{nk} = g_{kn} = -\int x^n x^k \exp\left[-\sum_{m=0}^{N} \lambda_m\,x^m\right] \mathrm{d}x = -G_{n+k}(\boldsymbol{\lambda}) \quad n,k = 0,\ldots,N \tag{11}$$

This means that the $[(N+1)\times(N+1)]$ matrix $\boldsymbol{G}$ in equation (7) becomes a symmetric Hankel matrix which is entirely defined by $2N+1$ values $G_n(\boldsymbol{\lambda}), n = 0,\ldots,2N$. So the algorithm in this case is the same as in the precedent one with two simplifications

1. In step 2 we do not need to write a seperate function to calculate the functions $\phi_n(x) = x^n, n = 0,\ldots,N$.

2. In step 4 the number of integral evaluations is reduced, because the elements g_{nk} of the matrix $\boldsymbol{G}$ are related to the integrals $G_n(\boldsymbol{\lambda})$ in equations (10). This matrix is defined entirely by only $2N+1$ components.

Case of trigonometrical moments

Another interesting special case is the case where the data are the Fourier components of $p(x)$

$$\mathrm{E}\left\{\exp\left(-jn\omega_0 x\right)\right\} = \int \exp\left(-jn\omega_0 x\right) p(x)\,\mathrm{d}x = \mu_n, \quad n = 0,\ldots,N, \tag{12}$$

where μ_n may be complex–valued and has the property $\mu_{-n} = \mu_n$. This means that we have the following relations

$$\begin{aligned} \phi_n(x) &= \exp\left(-jn\omega_0 x\right), \quad n = -N,\ldots,0,\ldots N, && (13)\\ p(x) &= \exp\left[-\mathrm{Real}\sum_{n=0}^{N} \lambda_n \exp\left(-jn\omega_0 x\right)\right], && (14)\\ G_n(\boldsymbol{\lambda}) &= \int \exp\left(-jn\omega_0 x\right) p(x)\,\mathrm{d}x, \quad n = 0,\ldots,N, && (15)\\ g_{nk} &= \begin{cases} -G_{n-k}(\boldsymbol{\lambda}) & \text{for } n \ge k, \\ -G_{n+k}(\boldsymbol{\lambda}) & \text{for } n < k \end{cases} \quad n,k = 0,\ldots,N, && (16) \end{aligned}$$

so that all the elements of the matrix $\boldsymbol{G}$ are related to the discrete Fourier transforms of $p(x)$. Note that $\boldsymbol{G}$ is a Hermitian Toeplitz matrix.

EXAMPLES AND NUMERICAL EXPERIMENTS

To illustrate the usefullness of the proposed programs we consider first the case of the Gamma distribution

$$p(x;\alpha,\beta) = \frac{\beta^{(1-\alpha)}}{\Gamma(1-\alpha)}\, x^{\alpha} \exp(-\beta x), \quad x > 0, \alpha < 1, \beta > 0. \tag{17}$$

This distribution can be considered as a ME distribution when the constraints are

$$\left\{\begin{array}{ll} \int p(x;\alpha,\beta)\,\mathrm{d}x & = 1 \\ \int x\, p(x;\alpha,\beta)\,\mathrm{d}x & = \mu_1 \\ \int \ln(x)\, p(x;\alpha,\beta)\,\mathrm{d}x & = \mu_2 \end{array}\right. \quad \left\{\begin{array}{lll} \text{normalization} & \phi_0(x) & = 1, \\ \phi_1(x) & & = x, \\ \phi_2(x) & & = \ln(x). \end{array}\right. \tag{18}$$

This is easy to verify because the equation (12) can be written as

$$p(x;\alpha,\beta) = \exp\left[-\lambda_0 - \lambda_1 x - \lambda_2 \ln(x)\right]$$

$$\text{with} \quad \lambda_0 = -\ln \frac{\beta^{(1-\alpha)}}{\Gamma(1-\alpha)}, \quad \lambda_1 = \beta \quad \text{and} \quad \lambda_2 = -\alpha.$$

Now consider the following problem

Given μ_1 and μ_2 determine λ_0, λ_1 and λ_2.

This can be done by the standard ME method. To do this, first we must define the range of x, `(xmin, xmax, dx)`, and write a function `fin_x` to calculate the functions $\phi_0(x) = 1$, $\phi_1(x) = x$ and $\phi_2(x) = \ln x$ (See the function `fin1_x` in Annex). Then we must define an initial estimate $\boldsymbol{\lambda}^0$ for $\boldsymbol{\lambda}$ and, finally, let the program works.

The case of the *Gamma* distribution is interesting because there is an analytic relation between (α,β) and the mean $m = \mathrm{E}\{x\}$ and variance $\sigma^2 = \mathrm{E}\{(x-m)^2\}$ which is

$$\begin{cases} m = (1-\alpha)/\beta \\ \sigma^2 = (1-\alpha)/\beta^2 \end{cases}, \tag{19}$$

or inversely

$$\begin{cases} \alpha = (\sigma^2 - m^2)/\sigma^2 \\ \beta = m/\sigma^2, \end{cases}, \tag{20}$$

so that we can use these relations to determine m and σ^2. Note also that the corresponding entropy of the final result is a byproduct of the function. Table (1) gives some numerical results obtained by ME_DENS1 program (See Annex).

Table 1.

μ_1	μ_2	α	β	m	σ^2
0.2000	-3.0000	0.2156	-3.0962	0.2533	0.0818
0.2000	-2.0000	-0.4124	-6.9968	0.2019	0.0289
0.3000	-1.5000	-0.6969	-5.3493	0.3172	0.0593

The next example is the case of a quartic distribution

$$p(x) = \exp\left[-\sum_{n=0}^{4} \lambda_n x^n\right]. \tag{21}$$

This distribution can be considered as a ME distribution when the constraints are

$$\mathrm{E}\{x^n\} = \int x^n p(x)\,\mathrm{d}x = \mu_n, \quad n = 0,\ldots,4 \quad \text{with} \quad \mu_0 = 1. \tag{22}$$

Now consider the following problem : Given $\mu_n, n = 1,\ldots,4$ calculate $\lambda_n, n = 0,\ldots,4$. This can be done by the ME_DENS2 program. Table (2) gives some numerical results obtained by this program:

Table 2.

μ_1	μ_2	μ_3	μ_4	λ_0	λ_1	λ_2	λ_3	λ_4
0	0.2	0.05	0.10	0.1992	1.7599	2.2229	-3.9375	0.4201
0	0.3	0.00	0.15	0.9392	0.000	-3.3414	0.0000	4.6875
0	0.3	0.00	0.15	0.9392	0.000	-3.3414	0.0000	4.6875

These examples show how to use the proposed programs. A third example is also given in Annex which shows how to use the ME_DENS3 program which considers the case of trigonometric moments.

CONCLUSIONS

In this paper we addressed first the class of ME distributions when the available data are a finite set of expectations $\mu_n = \mathrm{E}\{\phi_n(x)\}$ of some known functions $\phi_n(x)$, $n = 0, \ldots, N$. We proposed then three Matlab programs to solve this problem by a Newton–Raphson method in general case, in case of geometrical moments data where $\phi_n(x) = x^n$ and in case of trigonometrical moments where $\phi_n(x) = \exp(-jn\omega_0 x)$. Finally, we gave some numerical results for some special examples who show how to use the proposed programs.

REFERENCES

1. A. Zellnerr and R. Highfiled, “Calculation of Maximum Entropy Distributions and Approximation of Marginal Posterior Distributions”, *Journal of Econometrics* **37**, 1988, 195–209, North Holland.
2. D. Mukherjee and D.C. Hurst, “Maximum Entropy Revisited”, *Statistica Neerlandica* **38**, 1984, ną 1, 1–12.
3. Verdugo Lazo and P.N. Rathie, “On the Entropy of Continuous Probability Distributions”, *IEEE Trans.*, **vol. IT–24**, ną 1, 1978.
4. Gokhale, “Maximum Entropy Characterizations of some distributions”, *Statistical distributions in Scientific work*, **vol. 3**, 299–304 (G.P. Patil et al., Eds., Reidel, Dordrecht, Holland, 1975).
5. Jaynes, “Papers on probability, statistics and statistical physics”, *Reidel Publishing Company, Dordrecht*, Holland, 1983.
6. Matz, “Maximum Likelihood parameter estimation for the quartic exponential distributions”, *Technometrics*, **20**, 475–484, 1978.
7. Mohammad-Djafari A. et Demoment G., "Estimating Priors in Maximum Entropy Image Processing," *Proc. of ICASSP 1990*, pp: 2069-2072
8. Mohammad-Djafari A. et Idier J., "Maximum entropy prior laws of images and estimation of their parameters," *Proc. of The 10th Int. MaxEnt Workshop, Laramie, Wyoming*, published in Maximum-entropy and Bayesian methods, T.W. Grandy ed., 1990.

ANNEX A

```
function [lambda,p,entr]=me_dens1(mu,x,lambda0)
%ME_DENS1
%  [LAMBDA,P,ENTR]=ME_DENS1(MU,X,LAMBDA0)
%  This program calculates the Lagrange Multipliers of the ME
%  probability density functions p(x) from the knowledge of the
%  N contstraints in the form:
%  E{fin(x)}=MU(n)   n=0:N   with fi0(x)=1, MU(0)=1.
%
%  MU      is a table containing the constraints MU(n),n=1:N.
%  X       is a table defining the range of the variation of x.
%  LAMBDA0 is a table containing the first estimate of the LAMBDAs.
%          (This argument is optional.)
%  LAMBDA  is a table containing the resulting Lagrange parameters.
%  P       is a table containing the resulting pdf p(x).
%  ENTR    is a table containing the entropy values at each
%          iteration.
%
%  Author: A. Mohammad-Djafari
%  Date  : 10-01-1991
%
mu=mu(:); mu=[1;mu];            % add mu(0)=1
x=x(:); lx=length(x);           % x axis
xmin=x(1); xmax=x(lx); dx=x(2)-x(1);
%
if(nargin == 2)                 % initialize LAMBDA
 lambda=zeros(size(mu));              % This produces a uniform
 lambda(1)=log(xmax-xmin);      % distribution.
else
 lambda=lambda0(:);
end
N=length(lambda);
%
fin=fin1_x(x);                  % fin1_x(x) is an external
%                               % function which provides fin(x).
iter=0;
while 1                         % start iterations
  iter=iter+1;
  disp('---------------'); disp(['iter=',num2str(iter)]);
%
  p=exp(-(fin*lambda));         % Calculate p(x)
  plot(x,p);                    % plot it
%
  G=zeros(N,1);                 % Calculate Gn
  for n=1:N
   G(n)=dx*sum(fin(:,n).*p);
  end
%
  entr(iter)=lambda'*G(1:N);    % Calculate the entropy value
  disp(['Entropy=',num2str(entr(iter))])
%
  gnk=zeros(N,N);               % Calculate gnk
  gnk(1,:)=-G';  gnk(:,1)=-G;   % first line and first column
  for i=2:N                     % lower triangle part of the
   for j=2:i                    % matrix G
    gnk(i,j)=-dx*sum(fin(:,j).*fin(:,i).*p);
   end
  end
  for i=2:N                     % uper triangle part of the
   for j=i+1:N                  % matrix G
    gnk(i,j)=gnk(j,i);
   end
  end
%
  v=mu-G;                       % Calculate v
  delta=gnk\v;                  % Calculate delta
  lambda=lambda+delta;          % Calculate lambda
  eps=1e-6;                     % Stopping rules
  if(abs(delta./lambda)<eps),                      break, end
  if(iter>2)
   if(abs((entr(iter)-entr(iter-1))/entr(iter))<eps),break, end
  end
end
%
p=exp(-(fin*lambda));           % Calculate the final p(x)
plot(x,p);                      % plot it
entr=entr(:);
disp('-----  END  -------')
```

```
%----------------------------------
%ME1
%  This script shows how to use the function ME_DENS1
%  in the case of the Gamma distribution. (see Example 1.)
xmin=0.0001; xmax=1; dx=0.01;   % define the x axis
x=[xmin:dx:xmax]';
mu=[0.3,-1.5]';                 % define the mu values
[lambda,p,entr]=me_dens1(mu,x);
alpha=-lambda(3);   beta=lambda(2);
m=(1+alpha)/beta; sigma=m/beta;
disp([mu' alpha beta m sigma entr(length(entr))])
%----------------------------------
```

```
function fin=fin1_x(x);
% This is the external function which calculates
% the fin(x) in the special case of the Gamma distribution.
% This is to be used with ME_dens1.
  M=3;
  fin=zeros(length(x),M);
  fin(:,1)=ones(size(x));
  fin(:,2)=x;
  fin(:,3)=log(x);
return
```

```
function [lambda,p,entr]=me_dens2(mu,x,lambda0)
%ME_DENS2
%  [LAMBDA,P,ENTR]=ME_DENS2(MU,X,LAMBDA0)
%  This program calculates the Lagrange Multipliers of the ME
%  probability density functions p(x) from the knowledge of the
%  N moment contstraints in the form:
%  E{x^n}=mu(n)   n=0:N   with  mu(0)=1.
%
%  MU      is a table containing the constraints MU(n),n=1:N.
%  X       is a table defining the range of the variation of x.
%  LAMBDA0 is a table containing the first estimate of the LAMBDAs.
%          (This argument is optional.)
%  LAMBDA  is a table containing the resulting Lagrange parameters.
%  P       is a table containing the resulting pdf p(x).
%  ENTR    is a table containing the entropy values at each
%          iteration.
%
%  Author: A. Mohammad-Djafari
%  Date  : 10-01-1991
%
mu=mu(:); mu=[1;mu];            % add mu(0)=1
x=x(:); lx=length(x);           % x axis
xmin=x(1); xmax=x(lx); dx=x(2)-x(1);
%
if(nargin == 2)                 % initialize LAMBDA
 lambda=zeros(size(mu));              % This produces a uniform
 lambda(1)=log(xmax-xmin);      % distribution.
else
 lambda=lambda0(:);
end
N=length(lambda);
%
M=2*N-1;                        % Calcul de fin(x)=x.^n
fin=zeros(length(x),M);         %
fin(:,1)=ones(size(x));               % fi0(x)=1
for n=2:M
 fin(:,n)=x.*fin(:,n-1);
end
%
iter=0;
while 1                         % start iterations
  iter=iter+1;
  disp('---------------'); disp(['iter=',num2str(iter)]);
%
  p=exp(-(fin(:,1:N)*lambda));  % Calculate p(x)
  plot(x,p);                    % plot it
%
  G=zeros(M,1);                 % Calculate Gn
  for n=1:M
   G(n)=dx*sum(fin(:,n).*p);
  end
%
  entr(iter)=lambda'*G(1:N);    % Calculate the entropy value
  disp(['Entropy=',num2str(entr(iter))])
%
  gnk=zeros(N,N);               % Calculate gnk
  for i=1:N                     % Matrix G is a Hankel matrix
   gnk(:,i)=-G(i:N+i-1);
  end
%
  v=mu-G(1:N);                  % Calculate v
  delta=gnk\v;                  % Calculate delta
  lambda=lambda+delta;          % Calculate lambda
  eps=1e-6;                     % Stopping rules
  if(abs(delta./lambda)<eps),                        break, end
  if(iter>2)
   if(abs((entr(iter)-entr(iter-1))/entr(iter))<eps),break, end
  end
end
%
p=exp(-(fin(:,1:N)*lambda));    % Calculate the final p(x)
plot(x,p);                      % plot it
entr=entr(:);
disp('-----  END  -------')
end
```

```
%ME2
%  This script shows how to use the function ME_DENS2
%  in the case of the quartic distribution. (see Example 2.)
xmin=-1; xmax=1; dx=0.01;          % define the x axis
x=[xmin:dx:xmax]';
mu=[0.1,.3,0.1,.15]';               % define the mu values
[lambda,p,entr]=me_dens2(mu,x);
disp([mu;lambda;entr(length(entr))]')
```

```
function [lambda,p,entr]=me_dens3(mu,x,lambda0)
%ME_DENS3
%  [LAMBDA,P,ENTR]=ME_DENS3(MU,X,LAMBDA0)
%  This program calculates the Lagrange Multipliers of the ME
%  probability density functions p(x) from the knowledge of the
%  Fourier moments values :
%  E{exp[-j n w0 x]}=mu(n)   n=0:N   with  mu(0)=1.
%
%  MU      is a table containing the constraints MU(n),n=1:N.
%  X       is a table defining the range of the variation of x.
%  LAMBDA0 is a table containing the first estimate of the LAMBDAs.
%          (This argument is optional.)
%  LAMBDA  is a table containing the resulting Lagrange parameters.
%  P       is a table containing the resulting pdf p(x).
%  ENTR    is a table containing the entropy values at each
%          iteration.
%
% Author: A. Mohammad-Djafari
% Date  : 10-01-1991
%
mu=mu(:);mu=[1;mu];             % add mu(0)=1
x=x(:); lx=length(x);           % x axis
xmin=x(1); xmax=x(lx); dx=x(2)-x(1);
if(nargin == 2)                 % initialize LAMBDA
 lambda=zeros(size(mu));              % This produces a uniform
 lambda(1)=log(xmax-xmin);      % distribution.
else
 lambda=lambda0(:);
end
N=length(lambda);
%
M=2*N-1;                        % Calculate fin(x)=exp[-jnw0x]
fin=fin3_x(x,M);                % fin3_x(x) is an external
%                               % function which provides fin(x).
iter=0;
while 1                         % start iterations
 iter=iter+1;
 disp(’---------------’); disp([’iter=’,num2str(iter)]);
%
                                % Calculate p(x)
 p=exp(-real(fin(:,1:N))*real(lambda)+imag(fin(:,1:N))*imag(lambda));
 plot(x,p);                     % plot it
%
 G=zeros(M,1);                  % Calculate Gn
 for n=1:M
  G(n)=dx*sum(fin(:,n).*p);
 end
 %plot([real(G(1:N)),real(mu),imag(G(1:N)),imag(mu)])
%
 entr(iter)=lambda’*G(1:N);    % Calculate the entropy
 disp([’Entropy=’,num2str(entr(iter))])
%
 gnk=zeros(N,N);                % Calculate gnk
 for n=1:N                      % Matrix gnk is a Hermitian
  for k=1:n                     % Toeplitz matrix.
   gnk(n,k)=-G(n-k+1);          % Lower triangle part
  end
 end
 for n=1:N
  for k=n+1:N
   gnk(n,k)=-conj(G(k-n+1));    % Upper triangle part
  end
 end
%
 v=mu-G(1:N);                   % Calculate v
 delta=gnk\v;                   % Calculate delta
 lambda=lambda+delta;           % Calculate lambda
 eps=1e-3;                      % Stopping rules
 if(abs(delta)./abs(lambda)<eps),                   break, end
 if(iter>2)
  if(abs((entr(iter)-entr(iter-1))/entr(iter))<eps),break, end
 end
end
                                % Calculate p(x)
p=exp(-real(fin(:,1:N))*real(lambda)+imag(fin(:,1:N))*imag(lambda));
plot(x,p);                      % plot it
entr=entr(:);
disp(’-----  END  -------’)
end
```

```
%ME3
%  This scripts shows how to use the function ME_DENS3
%  in the case of the trigonometric moments.
clear;clf
xmin=-5; xmax=5; dx=0.5;         % define the x axis
x=[xmin:dx:xmax]';lx=length(x);
p=(1/sqrt(2*pi))*exp(-.5*(x.*x));% Gaussian distribution
plot(x,p);title('p(x)')
%
M=3;fin=fin3_x(x,M);            % Calculate fin(x)
%
mu=zeros(M,1);                  % Calculate mun
for n=1:M
  mu(n)=dx*sum(fin(:,n).*p);
end
%
w0=2*pi/(xmax-xmin);w=w0*[0:M-1]'; % Define the w axis
%
mu=mu(2:M);                     % Attention : mu(0) is added
                                % in ME_DENS3
[lambda,p,entr]=me_dens3(mu,x);
disp([mu;lambda;entr(length(entr))]')
```

```
function fin=fin3_x(x,M);
% This is the external function which calculates
% the fin(x) in the special case of the Fourier moments.
% This is to be used with ME_DENS3.
%
x=x(:); lx=length(x);           % x axis
xmin=x(1); xmax=x(lx); dx=x(2)-x(1);
%
fin=zeros(lx,M);                %
fin(:,1)=ones(size(x));               % fi0(x)=1
w0=2*pi/(xmax-xmin);jw0x=(sqrt(-1)*w0)*x;
for n=2:M
 fin(:,n)=exp(-(n-1)*jw0x);
end
return
```