MAXENTMC – A MAXIMUM ENTROPY ALGORITHM WITH MOMENT CONSTRAINTS

RAFAIL V. ABRAMOV

Contents

1. Introduction

MaxEntMC is a maximum entropy algorithm with moment constraints, which can be adapted to computation of the maximum entropy problem with the user-defined set of moment constraints, user-defined quadrature (and, therefore, domain) on the N-dimensional Euclidean space (where N is realistically no greater than 3 or 4 due to computational limitations on the numerical quadrature), via user-defined iterations of finding the critical point of the corresponding objective function (such as the Newton or quasi-Newton methods). In a certain sense, MaxEntMC is a set of tools to produce an individually tailored maximum entropy algorithm for a given problem, although some simple examples of maximum entropy quadrature and iteration implementations are included with the code. The MaxEntMC provides the user with the following core tools for the maximum entropy computation:

(1) A set of data structures and routines to initialize, save and load moment constraints and corresponding Lagrange multipliers of arbitrary dimension and power;
(2) A set of quadrature helper routines to implement a parallel (currently shared-memory multithreaded) numerical quadrature for density moments over a set of user-specified abscissas and weights;
(3) A set of routines to transform the computed density moments into the gradient vector and hessian matrix of the Lagrangian objective function.

In addition, MaxEntMC provides the user with the following example routines for better illustration:

(1) A simple uniform quadrature routine over a user-specified rectangular domain;

(2) A Newton method routine with a primitive inexact line search for computation of the critical point of the objective Lagrangian function.

It is, however, expected that the user will tailor the algorithm precisely to their particular maximum entropy problem by implementing their own quadrature and iteration methods.

The manual is organized as follows. Section 2 presents a mathematical formulation of the maximum entropy problem with moment constraints. Section 3.2 documents the routines necessary for operations with moment constraints and Lagrange multipliers. Section 3.3 documents the quadrature helper routines which are needed to set up the user-specified quadrature. Section 3.4 documents the routines which are used to extract the gradient and hessian of the Lagrangian function from the computed moments in a generic vector and matrix form.

2. Mathematical formulation of the maximum entropy problem with moment constraints

The content of this section is based on my paper titled *"The multidimensional maximum entropy moment problem: A review on numerical methods"*, published in Commun. Math. Sci. Some parts of the section content below may be copied, with minor changes, from that work. For more details, please refer to that work and references therein.

2.1. **Notations.** Let N be a positive integer number. Let $\mathbb{R}^N$ be an N-dimensional space of real numbers, and let $\mathbb{Z}^N$ be an N-dimensional space of nonnegative integer numbers. Let $\boldsymbol{x}$ denote an element of $\mathbb{R}^N$, and let $\boldsymbol{i}$ denote an element of $\mathbb{Z}^N$. We define a *monomial* $\boldsymbol{x}^{\boldsymbol{i}}$ as

$$\boldsymbol{x}^{\boldsymbol{i}} = x_1^{i_1} x_2^{i_2} \ldots x_N^{i_N} = \prod_{k=1}^{N} x_k^{i_k}. \tag{2.1}$$

We denote the *power* $|\boldsymbol{i}|$ of the monomial in (2.1) as

$$|\boldsymbol{i}| = i_1 + i_2 + \ldots + i_N = \sum_{k=1}^{N} i_k. \tag{2.2}$$

Let U be a domain in $\mathbb{R}^N$. Let $\rho : U \to \mathbb{R}$ denote any nonnegative continuous function on U, with the condition that

$$\int_U \rho(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} < \infty. \tag{2.3}$$

In this case, ρ is called the *density*. Given an element $\boldsymbol{i} \in \mathbb{Z}^N$, we denote the corresponding *moment* $m_{\boldsymbol{i}}[\rho]$ as

$$m_{\boldsymbol{i}}[\rho] = \int_U \boldsymbol{x}^{\boldsymbol{i}} \rho(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}. \tag{2.4}$$

Let P be a finite subset of $\mathbb{Z}^N$. Let $\rho_c : U \to \mathbb{R}$ denote any ρ for which all $m_{\boldsymbol{i}}$, such that $\boldsymbol{i} \in P$, are finite, and are given by a set of real numbers $c_{\boldsymbol{i}} \in \mathbb{R}$, for all $\boldsymbol{i} \in P$:

$$m_{\boldsymbol{i}}[\rho_c] = c_{\boldsymbol{i}}, \qquad \forall \boldsymbol{i} \in P. \tag{2.5}$$

It is then said that $\rho_{\boldsymbol{c}}$ possesses a set of moments (or *moment constraints*, in the context of what is presented below) $\boldsymbol{c}$. There are two things which need to be emphasized:

(1) For an arbitrarily specified set of real numbers $\boldsymbol{c}$, the corresponding density $\rho_{\boldsymbol{c}}$ does not have to exist (for example, suppose that $\boldsymbol{i}$ includes only even integers, yet $c_{\boldsymbol{i}} < 0$);
(2) When $\rho_{\boldsymbol{c}}$ nonetheless exists, it does not have to be unique.

What is presented below deals with the second situation.

2.2. **Maximum entropy under moment constraints.** We define the *Shannon entropy* $S[\rho]$ as

$$S[\rho] = -\int_U \rho(\boldsymbol{x}) \ln \rho(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}. \tag{2.6}$$

Shannon entropy is recognized as a measure of uncertainty in probability densities. In the context of the maximum entropy problem, the goal is to find $\rho_{\boldsymbol{c}}^*$ among all $\rho_{\boldsymbol{c}}$, such that is maximizes S:

$$\rho_{\boldsymbol{c}}^* = \arg\max_{\rho_{\boldsymbol{c}}} S[\rho_{\boldsymbol{c}}]. \tag{2.7}$$

It can be shown via variational analysis that $\rho_{\boldsymbol{c}}^*$ belongs to the family of functions $f_{\boldsymbol{\lambda}} : U \to \mathbb{R}$, where $\boldsymbol{\lambda} = \{\lambda_{\boldsymbol{i}}\}$ are real numbers (called the *Lagrange multipliers*), $\boldsymbol{i} \in P$. $f_{\boldsymbol{\lambda}}(\boldsymbol{x})$ is explicitly given by

$$f_{\boldsymbol{\lambda}}(\boldsymbol{x}) = \exp\left(\sum_{\boldsymbol{i}\in P} \lambda_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}\right). \tag{2.8}$$

Depending on the sets U, P and on the values $\boldsymbol{\lambda}$, $f_{\boldsymbol{\lambda}}(\boldsymbol{x})$ may or may not be a density; indeed, observe that while any $f_{\boldsymbol{\lambda}}$ is certainly nonnegative, its integral over U does not have to be finite. More precisely, any $f_{\boldsymbol{\lambda}}$ is a density over a finite domain U, but not all $f_{\boldsymbol{\lambda}}$ are densities when U is not a finite domain.

Our goal now is to find the set $\boldsymbol{\lambda}^*$ for which

$$f_{\boldsymbol{\lambda}^*}(\boldsymbol{x}) = \rho_{\boldsymbol{c}}^*(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in U. \tag{2.9}$$

This is accomplished by computing the minimum of the *Lagrangian function* (or shortly, the Lagrangian)

$$L(\boldsymbol{\lambda}) = m_{\boldsymbol{0}}[f_{\boldsymbol{\lambda}}] - \sum_{\boldsymbol{i}\in P} c_{\boldsymbol{i}} \lambda_{\boldsymbol{i}}. \tag{2.10}$$

The gradient (the vector of the first derivatives) and hessian (the matrix of the second derivatives) of the Lagrangian are given, respectively, via

$$\frac{\partial L}{\partial \lambda_{\boldsymbol{i}}} = m_{\boldsymbol{i}}[f_{\boldsymbol{\lambda}}] - c_{\boldsymbol{i}}, \tag{2.11a}$$

$$\frac{\partial^2 L}{\partial \lambda_{\boldsymbol{i}} \partial \lambda_{\boldsymbol{j}}} = m_{\boldsymbol{i}+\boldsymbol{j}}[f_{\boldsymbol{\lambda}}]. \tag{2.11b}$$

From the above expressions, it is easy to see that the gradient of L is zero when the moment constraints are met (that is, a critial point of L is found), and that the hessian of

L is positive definite, which means that the critical point is indeed a unique minimum. Note that, in general, the critial point does not have to exist.

The optimal set of Lagrange multipliers $\boldsymbol{\lambda}$ can be found iteratively, using a plethora of standard methods such as the Newton method, or a variable metric quasi-Newton method (such as the BFGS algorithm). The Newton method requires the computation of the hessian, while the quasi-Newton methods use a "finite difference" secant approximation for the hessian from the set of gradients computed at different points along the optimization path. Either iteration method can be implemented with MaxEntMC.

3. How to use MaxEntMC

Computation of the set of Lagrange multipliers $\boldsymbol{\lambda}$ from the set of constraints $\boldsymbol{c}$ can be split into the following tasks:

(1) Definition of constraints and Lagrange multipliers;
(2) Computation of moments from the Lagrange multipliers using the numerical quadrature;
(3) Conversion of computed moments into the value of the Lagrangian objective function, its gradient vector and hessian matrix;
(4) (Quasi-) Newton iterations over the set of Lagrange multipliers until suitable tolerance is achieved.

By the design of MaxEntMC, the last step is left to the user. Below we explain how to do other steps. All relevant data structures and routines, described below, are declared in `src/user/maxentmc.h`.

3.1. Basic types defined by MaxEntMC.

The two basic types defined by MaxEntMC are

(1) `maxentmc_index_t` – this is the integer type to hold the dimension of the problem and values of powers. It is currently defined as `uint8_t`, since it is highly unlikely that the dimension or power of the maximum entropy problem can realistically exceed 255.
(2) `maxentmc_float_t` – this is the floating point type used to hold real numbers in the algorithm. It is currently defined as `double`.

3.2. How to specify constraints and Lagrange multipliers.

The basic data type to hold a vector of real values with attached powers is given by

```
struct maxentmc_power_vector_struct {
    gsl_vector gsl_vec;     /* This is the standard GSL vector */
    struct maxentmc_power_struct * powers;   /* This is the pointer to the
         opaque structure which holds powers for each element of gsl_vec */
};
typedef struct maxentmc_power_vector_struct * maxentmc_power_vector_t;
  /* For convenience, the pointer to the structure is defined as a type */
```

Here, the values are contained `gsl_vec`, which is a standard GSL vector structure (with all GSL routines applicable to it, except for allocation/freeing), and are directly accessible by dereferencing `gsl_vec.data` pointer, which is of type `maxentmc_float_t`. On the other hand, powers is the pointer to the opaque structure which contains powers attached to each value contained in gsl_vec.

3.2.1. *Operations with an existing power vector.* When an object of type maxentmc_power_vector_t is already allocated and initialized (more on this later), the following basic operations are provided for it:

1.
```
void maxentmc_power_vector_free(maxentmc_power_vector_t v);
/* Frees the power vector v */
```

2.
```
maxentmc_index_t maxentmc_power_vector_get_dimension
                                 (maxentmc_power_vector_t v);
/* This function returns the dimension of the set of powers of the power
   vector v. On error, zero is returned */
```

3.
```
int maxentmc_power_vector_get_max_power(maxentmc_power_vector_t v,
                                        maxentmc_index_t * max_power);
/* Writes the maximum power of the power vector v into max_power.
   On error, -1 is returned, zero otherwise. */
```

4.
```
int maxentmc_power_vector_find_element(maxentmc_power_vector_t v,
                                       maxentmc_index_t p_1,
                                       ...
                                       maxentmc_index_t p_dim,
                                       size_t * pos);
/* Finds the position of the v->gsl_vec.data element corresponding to the
   prescribed set of powers p1,...,pdim, and writes it to the memory
   location pointed at by pos. Returns -1 on error, zero otherwise. */
```

5.
```
int maxentmc_power_vector_find_element_ca(maxentmc_power_vector_t v,
                                          maxentmc_index_t * p,
                                          size_t * pos);
/* The constant arity version of the previous function. The powers are
   specified in p[0],...,p[dim-1]. */
```

6.
```
int maxentmc_power_vector_get_powers(maxentmc_power_vector_t v, size_t pos,
                                     maxentmc_index_t * p_1,
                                     ...
                                     maxentmc_index_t * p_dim);
/* Writes the powers, corresponding to the element v->gsl_vec.data[pos],
   into memory locations pointed at by p1,...,pdim. Returns -1 on error,
   zero otherwise. */
```

7.
   ```
int maxentmc_power_vector_get_powers_ca(maxentmc_power_vector_t v,
                                        size_t pos, maxentmc_index_t * p);
/* The constant arity version of the previous function. The powers are
   written into p[0],...,p[dim-1]. */
```

8.
   ```
int maxentmc_power_vector_compute_polynomial(maxentmc_power_vector_t v,
                                             maxentmc_float_t x_1,
                                             ...
                                             maxentmc_float_t x_dim,
                                             maxentmc_float_t * result);
/* The values in v->gsl_vec.data are treated as coefficients of the
   polynomial with corresponding powers, and the value of the polynomial
   at the point x_1,...,x_dim is written into the memory location pointed at
   by result. Returns -1 on error, zero otherwise. */
```

9.
   ```
int maxentmc_power_vector_compute_polynomial_ca(maxentmc_power_vector_t v,
                                                maxentmc_float_t * x,
                                                maxentmc_float_t * result);
/* The constant arity version of the previous function. The point is
   given by x[0],...,x[dim-1]. */
```

10.
    ```
int maxentmc_power_vector_print(maxentmc_power_vector_t v, FILE * out);
/* Prints the contents of v into the stream out. Returns -1 on error,
   zero otherwise. */
```

11.
    ```
int maxentmc_power_vector_fwrite_values(maxentmc_power_vector_t v,
                                        FILE * out);
/* Writes the contents of v->gsl_vec.data into the stream out with
   host-to-network byte order conversion (currently, IEEE 754 floating
   point format is assumed). Returns -1 on error, zero otherwise. */
```

12.
    ```
int maxentmc_power_vector_fwrite_powers(maxentmc_power_vector_t v,
                                        FILE * out);
/* Writes the powers of v into the stream out with host-to-network byte
   order conversion. Returns -1 on error, zero otherwise. */
```

3.2.2. *Initialization of a power vector.* A power vector can be allocated and initialized (filled with data) in the following ways:

1. From an existing power vector. In this case, the new power vector inherits the powers from the existing power vector. The vector elements can then be initialized either by directly accessing the elements of `gsl_vec.data`, by reading previously saved data from a stream, or by arithmetic operations with vectors provided by GSL.

2. As a product of two existing power vectors (this is needed for the hessian computation). In this case, the new power vector obtains its powers from the product of powers of the two existing power vectors. The subsequent initialization of vector elements is same as above.
3. By reading the previously saved power structure from a stream. The subsequent initialization of vector elements is same as above.
4. From an object of the type `maxentmc_list_t`. This is the most general, safe (and slow) way of initializing a power vector.

Below is the list of allocation and initialization functions.

1.
```
maxentmc_vector_t maxentmc_power_vector_alloc(maxentmc_power_vector_t v);
/* Allocates a new vector, inheriting powers from v. The contents of
   gsl_vec.data are uninitialized upon return. Returns NULL on error. */
```

2.
```
maxentmc_vector_t maxentmc_power_vector_product_alloc
                                              (maxentmc_power_vector_t v1,
                                               maxentmc_power_vector_t v2);
/* Allocates a new vector with powers computed as the product of those
   of v1 and v2. The contents of gsl_vec.data are uninitialized upon
   return. Returns NULL on error. */
```

3.
```
maxentmc_vector_t maxentmc_power_vector_fread_powers(FILE * in);
/* Allocates a new vector, reading powers from the stream. The contents
   of gsl_vec.data are uninitialized upon return. Returns NULL on error.
   */
```

4.
```
maxentmc_vector_t maxentmc_power_vector_fread_values
                                   (maxentmc_power_vector_t v, FILE * in);
/* For an existing vector, reads the values for v->gsl_vec.data from the
   stream in. Returns -1 on error, zero otherwise. */
```

Update here.

3.3. How to compute moments.

To be written.

3.4. How to compute the gradient and hessian of the objective function.

To be written.