User manual for the multidimensional moment-constrained maximum entropy algorithm

Rafail Abramov*

1 Overview

The current manual provides user instructions for the multidimensional moment-constrained maximum entropy algorithm, which is described in [1] and [2] (**please refer to these publications if you use the code**). The complete algorithm consists of three datatypes, which are associated, naturally, with a set of input moments, its respective set of computed Lagrange multipliers, and a maximum entropy computational routine, which accepts an object of the moment datatype, and outputs the corresponding object of the Lagrange multiplier datatype. The general form of use for these three datatypes is the following:

1. The complete input information about the maximum entropy moment problem, that is, the dimension, maximum moment order, and values of all required moment constraints are specified in an object of the moment datatype, for which all appropriate routines are provided in each implemented language interface;
2. The initialized object of moment datatype is supplied to an object of the maximum entropy computational routine (for which a few computational parameters, such as the convergence tolerance and resolution of quadrature, can also be specified).
3. The maximum entropy computational routine writes calculated Lagrange multipliers into a provided object of the multiplier datatype. Then all the multipliers can be retrieved through the appropriate routines, which are provided for the multiplier datatype.

All three datatypes and associated routines are implemented for the C, C++ and FORTRAN computer languages. The following list of header files (for C and C++ programs) is included with the distribution:

---

*Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago, 851 S. Morgan st, Chicago, IL 60607. E-mail: abramov@math.uic.edu

◇ `maxent_c.h`

– the header file with structure definitions and routine declarations for the C implementation

◇ `maxent.h`

– the main header file for the C implementation. In order to use the algorithm, include this file into your source code via

```
#include<maxent.h>
```

◇ `maxent_cpp.h`

– the header file with class definitions for the C++ implementation

◇ `maxent.hh`

– the main header file for the C++ implementation. In order to use the algorithm, include this file into your source code via

```
#include<maxent.hh>
```

Additionally, several example programs for C, C++ and FORTRAN implementations which compute maximum entropy moment problems in the 1D and 2D settings are included.

2 Datatypes and routines

There are three datatypes which are involved in the computation process. These datatypes are associated with moments, multipliers, and the maximum entropy algorithm itself.

2.1 Datatype: moments

This datatype holds all the information pertaining to the moment constraints, namely the dimension of the domain N on which the moments are computed, the maximum power P of computed moments, and all the values of moments. The associated routines allow definition of the dimension and order of the datatype, and also reading and writing values of moments from and into the object of this datatype.

2.1.1 C moments declarations

```
moments m;
/* Declares object m of type moments */

moments moment_set_dim_pow (int N, int P);
/* Initializes to dimension N and power P. Returns the initialized
object */

void moment_free(moments m);
/* Frees previously initialized object */

int moment_get_dim (moments m);
/* Retrieves the dimension of a moment object */

int moment_get_pow (moments m);
/* Retrieves the total power of a moment object */

int moment_get_size (moments m);
/* Retrieves size (i.e. the total number of stored moments) of a
moment object */

int moment_set_value(moments m, int * p, double v);
/* Sets the moment with power combination p to the value v. p is an
integer array of length N, whose entries are set to powers of the
respective coordinates. For example, for N=2, p[0]=2, p[1]=3, v=2.5
the 2D-moment, corresponding to x^2y^3 is set to 2.5 */

double moment_get_value(moments m, int * p);
/* Retrieves the moment with power combination p. p is an integer
array of length N, whose entries are set to powers of the respective
coordinates. For example, for N=2, p[0]=2, p[1]=3 the returned value
corresponds to the moment with power combination x^2y^3 */

int moment_copy(moments m_from, moments m_to);
/* Copies the moment object from into moments object m_to. The object
m_to has to be initialized prior to this operation, but does not have
to be of the same dimension and/or power as m_from */

int moment_print (moments m, FILE * out);
/* Outputs the contents of m into the stream out in a human-readable
format. Setting out to stdout or stderr outputs the contents of m into
the C standard output or standard error stream, respectively */
```

2.1.2 C++ moments declarations

```
class moments
{
    moments(const int N, const int P);
    // Constructor, initializes moments object to dimension N and
    // power P. Omitting the parameters defaults to N=0, P=0

    moments(const moments & m);
    // Copy constructor

    ˜moments();
    // Destructor

    moments & set_dim_pow (const int N, const int P);
    // Sets the dimension of the moments object to N, and its power to P.
    // Returns the reference to itself

    int get_dim () const;
    // Retrieves the dimension of the moment object

    int get_pow () const;
    // Retrieves the power of the moment object

    int get_size () const;
    // Obtain size (i.e. the total number of stored moments) of a moment
    // object

    double & operator () (const int * p) const;
    // Used as rvalue, this operator retrieves the value of the moment
    // corresponding to the power combination in p. Used as lvalue, this
    // operator sets the value of the moment corresponding to the power
    // combination in p. For example, for N=2, p[0]=2, p[1]=3 corresponds
    // to the moment with power combination xˆ2yˆ3

    moments & operator = (const moments & m);
    // Assignment operator

    const moments & print(std::ostream & out) const;
    // Outputs the contents into the ostream out in a human-readable
    // format. Omitting the argument defaults to std::cout.  Returns the
    // reference to itself.
}
```

2.1.3 Fortran moments declarations

```
      INTEGER*8 M
C     Declares a moment object M (in effect, the pointer to the
C     corresponding structure will be stored in M). 8-byte integer
C     ensures enough space to store the address on 64-bit platforms.

      SUBROUTINE MOMENT_SET_DIM_POW(M,N,P)
C     Initializes M with dimension N and power P (N and P are integers)

      SUBROUTINE MOMENT_FREE(M)
C     Frees previously initialized M

      SUBROUTINE MOMENT_GET_DIM(M,n)
C     Writes the dimension of the moment object M into n (n is integer)

      SUBROUTINE MOMENT_GET_POW(M,p)
C     Writes the total power of the moment object M into p (p is integer)

      SUBROUTINE MOMENT_GET_SIZE(M,s)
C     Writes the total size (number of all stored moments) of the moment
C     object M into s (s is integer)

      SUBROUTINE MOMENT_GET_VALUE(M,p,v)
C     Retrieves the moment value corresponding to the power combination
C     in p (which is INTEGER array of length N) and writes it into v
C     (which is a DOUBLE PRECISION variable)

      SUBROUTINE MOMENT_SET_VALUE(M,p,v)
C     Sets the moment value corresponding to the power combination in p
C     (which is INTEGER array of length N) to the one provided in v
C     (which is a DOUBLE PRECISION variable)

      SUBROUTINE MOMENT_COPY(M,M2)
C     Copies the moment object from M to M2 (where M2 has to be
C     initialized, but not necessarily to the same dimension and power
C     as M)

      SUBROUTINE MOMENT_PRINT(M)
C     Outputs the contents of the moment object M into the standard output
C     file descriptor in a human-readable format.
```

2.2 Datatype: multipliers

This datatype holds all the information pertaining to the Lagrange multipliers, namely the dimension of the domain N on which the multipliers are computed, the maximum power P of monomials for computed multipliers, and all the values of multipliers. The associated routines allow definition of the dimension and order of the datatype, and also reading the values of multipliers from an object of this datatype.

2.2.1 C multipliers declarations

```
multipliers m;
/* Declares object m of type multipliers */

multipliers multiplier_init();
/* Initializes the multipliers object. Returns the initialized object */

void multiplier_free(multipliers m);
/* Frees previously initialized object */

int multiplier_get_dim (multipliers m);
/* Retrieves the dimension of a multiplier object */

int multiplier_get_pow (multipliers m);
/* Retrieves the total power of a multiplier object */

int multiplier_get_size (multipliers m);
/* Retrieves size (i.e. the total number of stored multipliers) of a
multiplier object */

double multiplier_get_value(multipliers m, int * p);
/* Retrieves the multiplier with power combination p. p is an integer
array of length N, whose entries are set to powers of the respective
coordinates. For example, for N=2, p[0]=2, p[1]=3 the returned value
corresponds to the multiplier with power combination x^2y^3 */

double multiplier_get_poly_value (multipliers m, double * x);
/* This operator retrieves the value of the polynomial at point x
comprised of products of multipliers with their corresponding
monomials. In the context of a maximum entropy probability density,
the returned value is the value of the argument of the PDF exponent at
the point x. Here x is an array of length N. */

int multiplier_copy(multipliers m_from, multipliers m_to);
```

```
/* Copies the multiplier object from into multipliers object m_to. The
object m_to has to be initialized prior to this operation, but does
not have to be of the same dimension and/or power as m_from */

int multiplier_print (multipliers m, FILE * out);
/* Outputs the contents of m into the stream out in a human-readable
format. Setting out to stdout or stderr outputs the contents of m into
the C standard output or standard error stream, respectively */
```

2.2.2 C++ multipliers declarations

```
class multipliers
{
    multipliers(const int N, const int P);
    // Constructor, initializes multipliers object to dimension N and
    // power P. Omitting the parameters defaults to N=0, P=0

    multipliers(const multipliers & m);
    // Copy constructor

    ~multipliers();
    // Destructor

    int get_dim () const;
    // Retrieves the dimension of the multiplier object

    int get_pow () const;
    // Retrieves the power of the multiplier object

    int get_size () const;
    // Obtain size (i.e. the total number of stored multipliers) of a
    // multiplier object

    double & operator () (const int * p) const;
    // This operator retrieves the value of the multiplier corresponding
    // to the power combination in p. For example, for N=2, p[0]=2,
    // p[1]=3 corresponds to the multiplier with power combination x^2y^3

    double & operator () (const double * x) const;
    // This operator retrieves the value of the polynomial at point x
    // comprised of products of multipliers with their corresponding
    // monomials. For a maximum entropy PDF, it is the argument in the
    // PDF exponent at the point x. x is an array of length N.
    multipliers & operator = (const multipliers & m);
    // Assignment operator

    const multipliers & print(std::ostream & out) const;
    // Outputs the contents into the ostream out in a human-readable
    // format. Omitting the argument defaults to std::cout.  Returns the
    // reference to itself.
}
```

2.2.3 Fortran multipliers declarations

```
      INTEGER*8 M
C     Declares a multiplier object M (in effect, the pointer to the
C     corresponding structure will be stored in M). 8-byte integer
C     ensures enough space to store the address on 64-bit platforms.

      SUBROUTINE MULTIPLIER_SET_DIM_POW(M,N,P)
C     Initializes M with dimension N and power P (N and P are integers)

      SUBROUTINE MULTIPLIER_FREE(M)
C     Frees previously initialized M

      SUBROUTINE MULTIPLIER_GET_DIM(M,n)
C     Writes the dimension of the multiplier object M into n (n is integer)

      SUBROUTINE MULTIPLIER_GET_POW(M,p)
C     Writes the total power of the multiplier object M into p (p is
C     integer)

      SUBROUTINE MULTIPLIER_GET_SIZE(M,s)
C     Writes the total size (number of all stored multipliers) of the
C     multiplier object M into s (s is integer)

      SUBROUTINE MULTIPLIER_GET_VALUE(M,p,v)
C     Retrieves the multiplier value corresponding to the power combination
C     in p (which is INTEGER array of length N) and writes it into v
C     (which is a DOUBLE PRECISION variable)

      SUBROUTINE MULTIPLIER_SET_VALUE(M,p,v)
C     Sets the multiplier value corresponding to the power combination in p
C     (which is INTEGER array of length N) to the one provided in v
C     (which is a DOUBLE PRECISION variable)
```

```
      SUBROUTINE MULTIPLIER_COPY(M,M2)
C     Copies the multiplier object from M to M2 (where M2 has to be
C     initialized, but not necessarily to the same dimension and power as
C     M)

      SUBROUTINE MULTIPLIER_PRINT(M)
C     Outputs the contents of the multiplier object M into the standard
C     output file descriptor in a human-readable format.
```

2.3 Datatype: maxent

This datatype holds all the information pertaining to the maximum entropy algorithm, namely the tolerance of the convergence algorithm (which is a gradient L_∞ norm), the order of the Gauss-Hermite quadrature, and all its abscissas and weights. The associated routines allow setting the convergence tolerance and order of the Gauss-Hermite quadrature, and computation of the maximum entropy problem by supplying the moment constraints in the form of a moment object. The computed Lagrange multipliers are written into the supplied multiplier object.

2.3.1 C maxent declarations

```
maxent m; /* Declares object of type maxent */

maxent maxent_set_quad_size (int s);
/* Initializes object of type maxent with Gauss-Hermite quadrature of
order s. The object is returned */

void maxent_free(maxent m);
/* Frees previously initialized object */

int maxent_set_tol(maxent m, double tol);
/* Sets convergence tolerance to tol. Default is 1E-06 */

int maxent_info(maxent m, int info);
/* If the argument info is nonzero, verbose output will be emitted to
stdout during computations of maximum entropy problems
thereafter. Calling this routine with info set to zero prevents
verbose output for consequent computations */

int maxent_compute(maxent m, moments mm, multipliers mt);
/* This routine attempts to compute the optimal set of Lagrange
multipliers, which are then stored in mt, from the moment constraints
```

```
provided in mm, using the maxent object m. The return value is zero if
no error has occured, and nonzero if there was an error. */
```

2.4 C++ maxent declarations

```
class maxent
{
    maxent(const int s);
    // Constructor with Gauss-Hermite quadrature order s. Omitting s
    // defaults to zero (i.e. the quadrature is defunct).

    ~maxent();
    // Destructor

    maxent & set_quad_size (const int s);
    // Sets the order of the Gauss-Hermite quadrature to s. Omitting s
    // defaults to zero (i.e. the quadrature is defunct). Returns the
    // reference to itself.

    maxent & set_tol(const double tol);
    // Sets convergence tolerance to tol. Default is 1E-06. Omitting
    // the argument does not change the current tolerance.

    maxent & info(const bool info);

    // If the argument info is true, verbose output will be emitted to
    // stdout during computations of maximum entropy problems
    // thereafter. Calling this routine with info set to false prevents
    // verbose output for consequent computations. Returns the reference
    // to itself.

    int operator () (const moments & mm, multipliers & m);
    // This routine attempts to compute the optimal set of Lagrange
    // multipliers, which are then stored in mt, from the moment
    // constraints provided in mm, using the maxent object m. The return
    // value is zero if no error has occured, and nonzero if there was an
    // error.
}
```

2.4.1 Fortran maxent declarations

```
      INTEGER*8 M
C     Declares a maxent object M (in effect, the pointer to the
C     corresponding structure will be stored in M). 8-byte integer
C     ensures enough space to store the address on 64-bit platforms.

      SUBROUTINE MAXENT_SET_QUAD_SIZE(M,s)
C     Initializes object M of type maxent with Gauss-Hermite quadrature of
C     order s (which is INTEGER).

      SUBROUTINE MAXENT_FREE(M)
C     Frees previously initialized object M

      SUBROUTINE MAXENT_SET_TOL(M,tol)
C     Sets convergence tolerance of the maxent object M to tol (which is
C     DOUBLE PRECISION). Default is 1E-06

      SUBROUTINE MAXENT_INFO(M,info)
C     If the argument info (which is of type INTEGER) is nonzero, verbose
C     output will be emitted to standard output descriptor during
C     computations of maximum entropy problems thereafter. Calling this
C     routine with info set to zero prevents verbose output for consequent
C     computations

      SUBROUTINE MAXENT_COMPUTE(M,MM,MT,S)
C     This routine attempts to compute the optimal set of Lagrange
C     multipliers, which are then stored in MT, from the moment constraints
C     provided in MM, using the maxent object M. The argument S (which is
C     of type INTEGER) contains the exit status of the computation (zero if
C     no errors have occured, and nonzero if errors were encountered)
```

References

[1] R. Abramov. A practical computational framework for the multidimensional moment-constrained maximum entropy principle. *J. Comp. Phys.*, 211(1):198–209, 2006.

[2] R. Abramov. An improved algorithm for the multidimensional moment-constrained maximum entropy problem. *J. Comp. Phys.*, 226:621–644, 2007.