`MAXENT-F90`: Fortran 90 Library for Maximum-Entropy Basis Functions[1]

User's Reference Manual
Version 1.4

N. Sukumar
UC Davis
E-mail: nsukumar@ucdavis.edu

June 27, 2008

---

[1]Software library released under the GNU LGPL version 3 license.

Contents

Abstract

This manual describes the Fortran 90 implementation of maximum-entropy basis functions. The main ingredients of the theory are presented, and then the numerical implementation is touched upon. Instructions on the installation and execution of the code, as well as on writing an interface to the library are presented. Each program module and the most important functions in each module are discussed. The F90 library can be used for different applications of maximum-entropy basis functions such as meshfree Galerkin methods and data approximation in lower- and higher-dimensional parameter spaces ($\mathbb{R}^d$, $d \geq 1$).

1 Information-Theoretic Entropy Approximants

Shannon [1] introduced the concept of entropy in information theory, with an eye on its applications in communication theory. The general form of informational entropy (Shannon-Jaynes or relative entropy functional) is [2–4]:

$$H(p, m) = -\sum_{i=1}^{n} p_i \ln\left(\frac{p_i}{m_i}\right) \quad \text{or} \quad H(p, m) = -\int p(x) \ln\left(\frac{p(x)}{m(x)}\right) dx, \qquad (1.1)$$

where m is a p-estimate (prior distribution). The quantity $D(p\|m) = -H(p, m)$ is also referred to as the Kullback-Leibler (KL) distance. As a means for least-biased statistical inference in the presence of testable constraints, Jaynes's used the Shannon entropy to propose the principle of maximum entropy [5], and if the KL-distance is adopted as the objective functional, the variational principle is known as the principle of minimum relative entropy [4].

Consider a set of distinct nodes in $\mathbb{R}^d$ that are located at x^i $(i = 1, 2, \ldots, n)$, with $D = \text{con}(x^1, \ldots, x^n) \subset \mathbb{R}^d$ denoting the convex hull of the nodal set. For a real-valued function $u(x) : D \to \mathbb{R}$, the numerical approximation for $u(x)$ is:

$$u^h(x) = \sum_{i=1}^{n} \phi_i(x) u_i, \qquad (1.2)$$

where $x \in D$, $\phi_i(x)$ is the basis function associated with node i, and u_i are coefficients. The use of basis functions that are constructed independent of an *underlying mesh* has become popular in the past decade—meshfree Galerkin methods are a common target application for such approximation schemes [6–10]. The construction of basis functions using information-theoretic variational principles is a new development [11–14]; see Reference [14] for a recent review on meshfree basis functions. To obtain basis functions using the maximum-entropy formalism, the Shannon entropy functional (*uniform prior*) and a modified entropy functional (*Gaussian prior*) were introduced in References [11] and [12], respectively, which was later generalized by adopting the Shannon-Jaynes entropy functional (*any prior*) [14]. The implementation of these new basis functions has been carried out, and this manual describes a Fortran 90 library for computing maximum-entropy (max-ent) basis functions and their first and second derivatives for any prior weight function.

We use the relative entropy functional given in Eq. (1.1) to construct max-ent basis functions. The variational formulation for maximum-entropy approximants is: find $x \mapsto \phi(x) : D \to \mathbb{R}_+^n$ as the solution of the following constrained (convex or concave with min or max, respectively) optimization problem:

$$\min_{\phi \in \mathbb{R}_+^n} -f(x; \phi) = \max_{\phi \in \mathbb{R}_+^n} f(x; \phi), \quad f(x; \phi) = -\sum_{i=1}^{n} \phi_i(x) \ln\left(\frac{\phi_i(x)}{w_i(x)}\right), \qquad (1.3a)$$

subject to the linear reproducing conditions:

$$\sum_{i=1}^{n} \phi_i(x) = 1, \tag{1.3b}$$

$$\sum_{i=1}^{n} \phi_i(x)(x^i - x) = 0, \tag{1.3c}$$

where $\mathbb{R}_+^n$ is the non-negative orthant, $w_i(x) : D \to \mathbb{R}_+$ is a non-negative weight function (*prior estimate* to ϕ_i), and the linear constraints form an under-determined system. On using the method of Lagrange multipliers, the solution of the variational problem is [14]:

$$\phi_i(x) = \frac{Z_i(x; \lambda)}{Z(x; \lambda)}, \quad Z_i(x; \lambda) = w_i(x) \exp(-\lambda \cdot \tilde{x}^i), \tag{1.4}$$

where $\tilde{x}^i = x^i - x$ $(x, x^i \in \mathbb{R}^d)$ are shifted nodal coordinates, $\lambda \in \mathbb{R}^d$ are the d Lagrange multipliers (implicitly dependent on the point x) associated with the constraints in Eq. (1.3c), and $Z(x) = \sum_j Z_j(x; \lambda)$ is known as the partition function in statistical mechanics. The smoothness of maximum-entropy basis functions for the Gaussian prior was established in Reference [12]; the continuity for any C^k $(k \geq 0)$ prior was proved in Reference [15].

2 Numerical Implementation

On considering the dual formulation, the solution for the Lagrange multipliers can be written as [16, 17]

$$\lambda^* = \arg\min F(\lambda), \quad F(\lambda) := \ln Z(\lambda), \tag{1.5}$$

where λ^* is the optimal solution that is desired. Since F is strictly convex in the interior of D, convex optimization algorithms (for example, Newton's method and families of gradient descent) are a natural choice. The steps in these algorithm are:

1. Start with iteration counter $k = 0$. The initial guess $\lambda^0 = 0$ and let ϵ be the desired convergence tolerance. For the convergence tolerance, $\epsilon = 10^{-14}$–10^{-10} is suitable (see sample input data files in the `tests` sub-directory);

2. Compute $g^k := \nabla_\lambda F(\lambda^k)$ (gradient of F) and $H^k := \nabla_\lambda \nabla_\lambda F(\lambda^k)$ (Hessian of F);

3. Determine a suitable search direction, $\Delta\lambda^k$. For steepest descent, $\Delta\lambda^k = -g^k$ and for Newton's method, $\Delta\lambda^k = -\left(H^k\right)^{-1} g^k$ (matrix-vector notation) are used;

4. Update: $\lambda^{k+1} = \lambda^k + \alpha\Delta\lambda^k$, where α is the step size. For steepest descent, a variable step size (line search) algorithm, which is presented in Reference [18], is used to determine α, and for Newton's method (*damped* or *guarded*), the line search is used to set the step size if the error is greater than 10^{-4} and otherwise, $\alpha = 1$ is used;

5. Check convergence: if $|g^{k+1}| > \epsilon$, increment the iteration counter, $k \leftarrow k + 1$, and goto 2, else continue;

6. Set $\lambda^* = \lambda^{k+1}$ and compute the max-ent basis functions using Eq. (1.4).

3   Installing and Testing the Code

Unpack the tar archive on a UNIX machine using the command

```
tar -xvf maxent.tar
```

The archive consists of the main directory `MAXENT-V1.4` and several sub-directories. Use the command `cd MAXENT-V1.4` to change to the `MAXENT-V1.4` directory. Now, you will see the following sub-directories and files:

- `README`: README file that points to this document for instructions on the installation and use of the library;

- `COPYING`: License file that contains the GNU GPL version 3 license;

- `COPYING.LESSER`: License file that contains the GNU LGPL version 3 license;

- `bin`: executable is placed here after compiling and linking the code;

- `doc`: user's manual (PDF) is available in the `doc/manual` directory;

- `lib`: static library file `libmaxent.a` is placed here when `make maxentlib` is executed;

- `makefile`: a `makefile` at the top directory that depending on the desired action, executes the `makefile` in the sub-directory `src`;

- `makefile.inc`: file that is included in `makefile`; contains the settings for the compilers, compiler-options, and flags;

- `run`: a directory from where the program can be executed;

- `src`: all source files are located in this directory—two Fortran 90 module files (`maxent.f90` and `priorweightfunction.f90`) and the public domain Fortran 77 program `lbfgs.f` [19]. The subroutines `dpotrf` and `dpotri` from the `Lapack` library are used; linking to the `blas` and `lapack` libraries is required to have access to these subroutines. These subroutines are used only if $d > 3$. If data

3

approximation in 1-, 2-, or 3-dimensions is to be performed, these libraries are not needed. If so, set the `DEFS` macro to be empty in the file `makefile.inc` in the `src` directory; also see the `makefile` that resides in this sub-directory.

- `tests`: sample input data files are provided; the outputs that are generated when these input files are used in the program execution are available in the `tests/results` directory;

3.1 Compiling and linking

If you have changed directories, go back to the `MAXENT-V1.4` directory. Four options are available in the `makefile` in this directory:

1. Typing `make` or `make maxent` will result in transfer of control to the `makefile` in the `src` directory, where `make maxent` will be executed. The g95 Fortran 90 compiler/linker and the `g77` (GNU Fortran 77 compiler) are used. If all goes well in the compiling and linking phases, the executable file, `bin/maxent`, will be created.

2. Typing `make maxentlib` will result in transfer of control to the `makefile` in the `src` directory, where `make maxentlib` will be executed, and if all goes well in the compiling phase, the library file, `lib/libmaxent.a`, will be created.

3. Typing `make check` will result in the execution (by default, output will be sent to the terminal) of all the example input files in the `tests` sub-directory.

4. Typing `make clean` will result in the deletion of all object files, mod files, library file, and the executable.

3.2 Testing the code using a sample input file

If the executable (`bin/maxent`) has been installed by invoking `make` or `make maxent` in the top-directory, then one can run the program. A few sample input data files are provided in the `tests` sub-directory. All the input data files can be run in the top-directory by executing:

```
make check > <output-file>
```

and the output is piped to the `<output-file>`. If you would like to run individual test examples, then use the command `cd run` to change to the `run` directory. To execute the program from this directory, you have to invoke the following command:

```
../bin/maxent < <path-to-input-data-file> > <output-file>
```

For example, if the input data file `tests/8nodes3d.dat` is used, then execute the following command:

```
../bin/maxent < ../tests/8nodes3d.dat > 8nodes3d-output.dat
```

A file `check.dat` is created, where the input parameters are printed out. The sample

input data file `tests/8nodes3d.dat` is reproduced below.

```
              ------------------------------------
              Input data file: tests/8nodes3d.dat
              ------------------------------------
    NSD and NODES
    3 8
    POINT
    0.25 0.15 0.05
    COORDS
    0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00
    1.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00
    0.0000000000000e+00  1.0000000000000e+00  0.0000000000000e+00
    1.0000000000000e+00  1.0000000000000e+00  0.0000000000000e+00
    0.0000000000000e+00  0.0000000000000e+00  1.0000000000000e+00
    1.0000000000000e+00  0.0000000000000e+00  1.0000000000000e+00
    0.0000000000000e+00  1.0000000000000e+00  1.0000000000000e+00
    1.0000000000000e+00  1.0000000000000e+00  1.0000000000000e+00
    MAXITER AND TOL
    100 1.d-14
    RMAX AND PRIOR AND SOLUTION-SCHEME
    1 uniform newton
```

All lines with non-numeric data are not read, but to facilitate understanding of the input data file it is advisable to indicate on these lines what parameters are expected to be read on the very next line. Referring to the above input data file, on the first line, `NSD` and `NODES` are the number of spatial dimensions and number of nodes, respectively. On the next line, `NSD` = 3 and `NODES` = 8 are indicated. On the third line, `POINT` refers to the Cartesian coordinates of the point in `NSD`-dimensions (for example, a Gauss integration point in a meshfree computation or an output point in a data approximation problem) where the evaluation of the maximum-entropy basis functions is needed. On the fourth line, `POINT` = {0.25,0.15,0.05} is indicated. On the fifth line, `COORDS` appears, which indicates that in the next 8 (`NODES` = 8) lines, the coordinates of the nodes are specified. In the above case, the nodal coordinates of a unit cube are specified. On line 14, `MAXITER` (maximum number of iterations) and `TOL` (convergence tolerance) are listed, and these values are set on line 15. On line 16, `RMAX`, `PRIOR`, and `SOLUTION-SCHEME` are indicated, and one particular setting for these parameters is shown on line 17. `RMAX` is the support size if a compactly-supported weight function is used as a prior. This value must be picked so that the `POINT` has at least `NSD+1` neighbors. There are five options that are available for the choice of a prior: `PRIOR` = {*uniform, cubic, quartic, gaussian, gaussian-rbf*}. For a *uniform* prior, `RMAX` should be set to the characteristic linear dimension of the domain. It is assumed that for any non-uniform prior, all the nodes listed in the input file are neighbors. Hence, do ensure that the value of the support size is appropriately set, so that for $x \in D$, $w_i(x) > 0$ for all i. Three different solution schemes

5

are available to solve the convex optimization problem (steepest descent, Newton's method, and **L**imited-memory **B**royden-**F**letcher-**G**oldfarb-**S**hanno (L-BFGS) algorithm): `SOLUTION-SCHEME` $= \{descent,\ newton,\ lbfgs\}$.

The output file `8nodes3d-output.dat` and `tests/results/8nodes3d.output` must match; the latter is shown below (in the interest of space, only ϕ_i are listed). For the input file `tests/3nodes1d-uniform.dat`, the exact solution for the max-ent basis functions is presented in Reference [14, Eq. (31)]. In the `tests` directory, in addition to 1D, 2D, and 3D data, an input file with nodes at the vertices of a hypercube in four dimensions ($D = [0, 1]^4$) is also provided. You can repeat the above checks for all the input data files that are available in the `tests` sub-directory to ensure that the installation and program output are correct.

```
   ------------------------------------------
   Output file: tests/results/8nodes3d.output
   ------------------------------------------


   .  .  .  .
   ITER and ERROR:  0 0.6224949798994366
   ITER and ERROR:  1 0.06695084055654463
   ITER and ERROR:  2 0.006905786017980535
   ITER and ERROR:  3 0.00017430863425530217
   ITER and ERROR:  4 5.2009267516942105E-8
   ITER and ERROR:  5 1.0552434222233532E-14


   ******************************************************
   **************** NEWTON METHOD *********************
   ******************************************************
   POINT =  0.25 0.15 0.05
   CONVERGENCE ATTAINED IN ITERATIONS =   6
   ASKING TOLERANCE =   1.E-14
   ERROR =  3.2148521626575033E-17
   LAGRANGE MULTIPLIERS
       1  1.0986122886681E+00
       2  1.7346010553881E+00
       3  2.9444389791664E+00
   BASISFUNCTIONS
       1 6.0562500000000E-01
       2 2.0187500000000E-01
       3 1.0687500000000E-01
       4 3.5625000000000E-02
       5 3.1875000000000E-02
       6 1.0625000000000E-02
       7 5.6250000000000E-03
       8 1.8750000000000E-03
```

3.3 Linking external programs with the max-ent library

If one wants to use the max-ent library with any other program, then first create the static library file `libmaxent.a` by invoking `make maxentlib` in the top-directory. Let the environment variable `MAXENTDIR` point to where `MAXENT-V1.4` is located. The library file is at `$(MAXENTDIR)/MAXENT-V1.4/lib/libmaxent.a`. To link this library file with any other Fortran 90 program, the following command can be used:

```
g95 -o a.out <object-files> -L $(MAXENTDIR)/MAXENT-V1.4/lib -lmaxent
```

If an interface from a C or C++ program is required, the appropriate call (depending on the compiler and the operating system) should be invoked. The subroutine call from a Fortran 90 program to compute the max-ent basis functions is given below (see `main.f90` and also `maxent.f90` for further details):

```
call drivermaxent(n,nsd,scheme,prior,xyz,p,rmax,D,maxit,eps,printflag,errorflag,
                  lengthscale,dettol,phi,dphi,ddphi)
```

The input parameters in the above subroutine are as follows:

1. `n` : number of neighbors of point p (a positive integer);

2. `nsd` : spatial dimension (a positive integer);

3. `scheme` : either *descent*, *newton*, or *lbfgs* (a string);

4. `prior` : either *uniform*, *cubic*, *quartic*, *gaussian*, or *gaussian-rbf* (a string);

5. `xyz` : nodal coordinates of the n neighbors (double precision array *xyz(n,nsd)*);

6. `p` : coordinates of the point p (double precision array *p(nsd)*);

7. `rmax` : support sizes of the n neighbors (double precision array *rmax(n)*);

8. `D` : anisotropic metric (double precision array *D(nsd,nsd,n)*);

9. `maxit` : maximum number of iteration (a positive integer);

10. `eps` : convergence tolerance (double precision);

11. `printflag` : print convergence information (logical—*.true.* or *.false.*);

12. `errorflag` : error flag (0 : pass, 1 : Hessian too small; 2 : non-convergence in `maxit` iterations);

13. `lengthscale` : length scale for domain (`optional argument`) so that convergence criterion is independent of the magnitude of nodal coordinates;

14. `dettol` : determinant tolerance (`optional argument`) with default of 10^{-16}

The output variables (`phi`, `dphi`, and `ddphi`) in the parameter-list are defined as `optional`:

1. `phi` : basis function vector (double precision *phi(n)*);

2. `dphi` : 1st derivatives of basis functions (double precision *dphi(nsd,n)*);

3. `ddphi` : 2nd derivatives of basis functions (double precision *ddphi(nsd*nsd,n)*);

Please note that any calling subroutine must <u>first</u> determine for every point p, the number of neighbors n for p, assign the nodal coordinates of these n nodes to the array *xyz*, and the support sizes of these n nodes must be assigned to the array *rmax*. Then only can a call to the subroutine `drivermaxent(...)` be invoked. It is assumed that computations for `dphi` and/or `ddphi` are requested in conjunction with `phi`. Hence, in the parameter-list, `phi` (ϕ_i is computed), `phi` and `dphi` (ϕ_i and $\nabla\phi_i$ are computed), `phi` and `ddphi` (ϕ_i and $\nabla\nabla\phi_i$ are computed), or all three (ϕ_i, $\nabla\phi_i$ and $\nabla\nabla\phi_i$ are computed) are valid.

4 Fortran 90 Source Files

There are two Fortran 90 modules that are part of the max-ent library. These are in the source files `maxent.f90` and `priorweightfunction.f90`, and implementations for *steepest descent* and *Newton's method* are available. To provide access to the L-BFGS algorithm, a driver subroutine is included in `maxent.f90`, with a call to the public-domain limited-memory BFGS subroutine (Fortran 77 implementation) [19]. If a different convex optimization algorithm is desired, the appropriate driver routine can be added in `maxent.f90` to provide an interface to the subroutine/function in which the algorithm of interest is implemented. We now discuss some of the main features and computational details that are contained in the two Fortran 90 modules.

4.1 Maximum-entropy module

In `maxent.f90`, all subroutines and functions are contained within the `maxent module`. The basis functions are computed in `function phimaxent()`, where Eq. (1.4) is used. To this end, the Lagrange multipliers are determined using the convex optimization algorithms that are presented in Section 2. Most of the functions and subroutines in this module are self-explanatory. We elaborate on the computation of the gradient of $F(\lambda) := \ln Z(\lambda)$, and also on the Hessian H, which is required in the Newton method. On taking the gradient (with respect to λ) of $F(\lambda)$, we obtain the negative of the left-hand side of the linear reproducing conditions in Eq. (1.3c):

$$g = -\sum_{i=1}^{n} \phi_i \tilde{x}^i, \tag{1.6}$$

where $\tilde{x}^i = x^i - x$ $(i = 1, 2, \ldots, n)$. This is coded in `function dfunc()`. The expressions for the Hessian of $F(\lambda)$ are given in Reference [11] (matrix form) as well as in the Appendix of Reference [12]. By definition, the components of the Hessian are:

$$H_{rs}(\lambda) = \frac{\partial^2 F(\lambda)}{\partial \lambda_r \partial \lambda_s}, \quad (r, s = 1, \ldots, d). \tag{1.7}$$

The Hessian matrix (see Eq. (45) in Reference [11]) in three dimensions $(d = 3)$ is:

$$H = \begin{bmatrix} <\tilde{x}_1^2> - <\tilde{x}_1>^2 & <\tilde{x}_1\tilde{x}_2> - <\tilde{x}_1><\tilde{x}_2> & <\tilde{x}_1\tilde{x}_3> - <\tilde{x}_1><\tilde{x}_3> \\ <\tilde{x}_1\tilde{x}_2> - <\tilde{x}_1><\tilde{x}_2> & <\tilde{x}_2^2> - <\tilde{x}_2>^2 & <\tilde{x}_2\tilde{x}_3> - <\tilde{x}_2><\tilde{x}_3> \\ <\tilde{x}_1\tilde{x}_3> - <\tilde{x}_1><\tilde{x}_3> & <\tilde{x}_2\tilde{x}_3> - <\tilde{x}_2><\tilde{x}_3> & <\tilde{x}_3^2> - <\tilde{x}_3>^2 \end{bmatrix} \tag{1.8}$$

where $< \cdot >$ is the expectation operator, which for a scalar-valued function $u(x)$ is:

$$<u(x)> = \sum_{i=1}^{n} \phi_i(x) u_i, \quad u_i = u(x^i). \tag{1.9}$$

The Hessian matrix and its inverse are computed in `function hessian(flag)` (`flag` is an optional Boolean argument) and `function invhessian(flag)`, respectively.

Let $\lambda = \lambda^*$ denote the converged solution for the Lagrange multipliers and ϕ_i^* the corresponding basis function solution for the ith node. Since $<\tilde{x}_r>^* = 0$ $(r = 1\text{–}3)$, the Hessian (`hessian(.true.)` is the call) is

$$H^* = \begin{bmatrix} <\tilde{x}_1^2>^* & <\tilde{x}_1\tilde{x}_2>^* & <\tilde{x}_1\tilde{x}_3>^* \\ <\tilde{x}_1\tilde{x}_2>^* & <\tilde{x}_2^2>^* & <\tilde{x}_2\tilde{x}_3>^* \\ <\tilde{x}_1\tilde{x}_3>^* & <\tilde{x}_2\tilde{x}_3>^* & <\tilde{x}_3^2>^* \end{bmatrix}, \quad H^* = \sum_{k=1}^{n} \phi_k^* \tilde{x}^k \otimes \tilde{x}^k. \tag{1.10}$$

The expressions for the derivatives of the basis functions are given below. We adopt the notations and approach presented in Arroyo and Ortiz [12]; in the interest of space, just the final results are indicated. We can write Eq. (1.4) as

$$\phi_i^*(x; \lambda^*) = \frac{\exp[f_i^*(x; \lambda^*)]}{\sum_{j=1}^{n} \exp[f_j^*(x; \lambda^*)]}, \quad f_i^*(x; \lambda^*) = \ln w_i(x) - \lambda^* \cdot \tilde{x}^i, \tag{1.11}$$

where λ^* is implicitly dependent on x. On using Eq. (1.11), we have

$$\nabla \phi_i^* = \phi_i^* \left(\nabla f_i^* - \sum_{j=1}^{n} \phi_j^* \nabla f_j^* \right), \tag{1.12a}$$

where ∇f_i^* is given by

$$\nabla f_i^* = \frac{\nabla w_i}{w_i} + \lambda^* + \tilde{x}^i \cdot [(H^*)^{-1} - (H^*)^{-1} \cdot A^*], \quad A^* = \sum_{k=1}^{n} \phi_k^* \tilde{x}^k \otimes \frac{\nabla w_k}{w_k}, \tag{1.12b}$$

9

and therefore the gradient of ϕ_i^* is

$$\nabla \phi_i^* = \phi_i^* \left\{ \tilde{x}^i \cdot \left[(H^*)^{-1} - (H^*)^{-1} \cdot A^* \right] + \frac{\nabla w_i}{w_i} - \sum_{j=1}^{n} \phi_j^* \frac{\nabla w_j}{w_j} \right\}. \tag{1.13}$$

If the prior $w_i(x)$ is a Gaussian radial basis function (see Reference [13]), then $w_i(x) = \exp(-\beta |x^i - x|^2)$ and Eq. (1.13) reduces to $\nabla \phi_i^* = \phi_i^* (H^*)^{-1} \cdot \tilde{x}^i$! This result appears in the Appendix of Reference [12]. In general, $\nabla \phi_i^*$ depends on ∇w_i through the expression given in Eq. (1.13). The gradient of the basis functions is computed in `function dphimaxent()`.

On taking the gradient of Eq. (1.13), we obtain the following expression for the second derivatives of the max-ent basis functions:

$$\begin{aligned}
\nabla \nabla \phi_i^* = {} & \frac{\nabla \phi_i^* \otimes \nabla \phi_i^*}{\phi_i^*} \\
& + \phi_i^* \left\{ -(H^*)^{-1} + \tilde{x}^i \cdot \nabla \big((H^*)^{-1} \big) + A^{*T} \cdot (H^*)^{-1} \right\} \\
& - \phi_i^* \left\{ \tilde{x}^i \cdot \nabla \big((H^*)^{-1} \big) \cdot A^* + \tilde{x}^i \cdot (H^*)^{-1} \cdot \nabla A^* \right\} \\
& + \phi_i^* \left\{ \nabla \left(\frac{\nabla w_i}{w_i} \right) - \sum_{j=1}^{n} \frac{\nabla w_j}{w_j} \otimes \nabla \phi_j^* - \sum_{j=1}^{n} \phi_j^* \nabla \left(\frac{\nabla w_j}{w_j} \right) \right\},
\end{aligned} \tag{1.14}$$

where on using the identity $H^* \cdot (H^*)^{-1} = I$, the gradient of the inverse of the Hessian can be written as

$$\nabla \big((H^*)^{-1} \big) = -(H^*)^{-1} \cdot \nabla H^* \cdot (H^*)^{-1}, \quad \nabla H^* = \sum_{k=1}^{n} \tilde{x}^k \otimes \tilde{x}^k \otimes \nabla \phi_k^*, \tag{1.15}$$

and therefore

$$\tilde{x}^i \cdot \nabla \big((H^*)^{-1} \big) = -\tilde{x}^i \cdot \left(\sum_{k=1}^{n} \tilde{v}^k \otimes \tilde{v}^k \otimes \nabla \phi_k^* \right), \quad \tilde{v}^k = \tilde{x}^k \cdot (H^*)^{-1}. \tag{1.16}$$

The term $-\tilde{x}^i \cdot \nabla \big((H^*)^{-1} \big) \cdot A^*$ in Eq. (1.14) is given by

$$-\tilde{x}^i \cdot \nabla \big((H^*)^{-1} \big) \cdot A^* = \tilde{x}^i \cdot \left(\sum_{k=1}^{n} \tilde{v}^k \otimes \tilde{w}^k \otimes \nabla \phi_k^* \right), \quad \tilde{w}^k = \tilde{v}^k \cdot A^*, \tag{1.17}$$

and the term $-\tilde{x}^i \cdot (H^*)^{-1} \cdot \nabla A^*$ is:

$$-\tilde{x}^i \cdot (H^*)^{-1} \cdot \nabla A^* = -\tilde{v}^i \cdot A_1^* + A_2^* \otimes \tilde{v}^i - \tilde{v}^i \cdot A_3^*, \tag{1.18a}$$

where

$$A_1^* = \sum_{k=1}^{n} \tilde{x}^k \otimes \frac{\nabla w_k}{w_k} \otimes \nabla \phi_k^*, \ A_2^* = \sum_{k=1}^{n} \phi_k^* \frac{\nabla w_k}{w_k}, \ A_3^* = \sum_{k=1}^{n} \phi_k^* \tilde{x}^k \otimes \nabla \left(\frac{\nabla w_k}{w_k} \right). \tag{1.18b}$$

The Hessian of the basis functions is computed in `function ddphimaxent()`. It satisfies the conditions $\sum_i \nabla \nabla \phi_i^* = 0$ and $\sum_i \nabla \nabla \phi_i^* \otimes \tilde{x}^i = 0$.

4.2 Prior weight function module

In `priorweightfunction.f90`, all subroutines and functions are contained within the `priorweightfunction module`. The radius of support for the nodal weight functions, metric D, and the *prior* are set in `subroutine setrmaxandpriorweight(...)`. This is done via a call to this function from an external program or from `subroutine drivermaxent(...)`, which is contained in the `maxent module`. The main program, `main.f90`, serves as an illustrative reference. The variables `rmax`, D, and `prior` are private variables, and hence once they are set, all functions within the module can access them. The weight functions are defined in `function weightfunction(`k,$\tilde{x}^k$,$D_k\tilde{x}^k$,q`)`. Here, k runs from 1 to n, $\tilde{x}^k := x^k - x$, and $D_k \neq I$ renders the nodal weight function support to be anisotropic. The variable $q := \sqrt{\tilde{x}^k D_k \tilde{x}^k}/\rho_{\max}^k$, where $\rho_{\max}^k$ is the radius of support of the nodal weight function. The following weight functions are available:

1. *uniform*: a constant weight function, i.e., $w(q) = 1$.

2. *cubic*: the weight function is a C^2 cubic spline function, which is given by

$$w(q) = \begin{cases} \dfrac{2}{3} - 4q^2 + 4q^3 & \text{if } 0 \leq q \leq \dfrac{1}{2}, \\ \dfrac{4}{3} - 4q + 4q^2 - \dfrac{4q^3}{3} & \text{if } \dfrac{1}{2} < q \leq 1, \\ 0 & \text{otherwise} \end{cases} \quad (1.19)$$

3. *quartic*: the weight function is a C^2 quartic polynomial function, which is given by

$$w(q) = \begin{cases} 1 - 6q^2 + 8q^3 - 3q^4 & \text{if } 0 \leq q \leq 1, \\ 0 & \text{otherwise} \end{cases} \quad (1.20)$$

4. *gaussian*: the weight function is a C^∞ Gaussian function, which is given by

$$w(q) = \begin{cases} \exp\left(-\dfrac{1}{1-q^2}\right) & \text{if } 0 \leq q \leq 1, \\ 0 & \text{otherwise} \end{cases} \quad (1.21)$$

5. *gaussian-rbf*: a C^∞ Gaussian radial basis function, $w(r) = \exp\left(-\beta r^2\right)$ [12], where $\beta \equiv \rho_{\max}^k$ is a constant for each node.

If a different prior is desired, it can be added in `function weightfunction(...)`. The gradient is defined in `function dweightfunction(...)`, and the Hessian of the weight function is defined in `function ddweightfunction(...)`.

5 Acknowledgements and Credits

Bibliography

[1] C. E. Shannon. A mathematical theory of communication. *The Bell Systems Technical Journal*, 27:379–423, 1948.

[2] S. Kullback. *Information Theory and Statistics*. Wiley, New York, NY, 1959.

[3] E. T. Jaynes. Information theory and statistical mechanics. In K. Ford, editor, *Statistical Physics: The 1962 Brandeis Lectures*, pages 181–218, New York, 1963. W. A. Benjamin.

[4] J. E. Shore and R. W. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, 26(1):26–36, 1980.

[5] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4):620–630, 1957.

[6] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139:3–47, 1996.

[7] S. N. Atluri and S. Shen. *The Meshless Local Petrov-Galerkin (MLPG) Method*. Tech Science Press, Encino, CA, 2002.

[8] G. R. Liu. *Mesh Free Methods: Moving Beyond the Finite Element Method*. CRC Press, Boca Raton, FL, 2003.

[9] S. Li and W. K. Liu. *Meshfree Particle Methods*. Springer-Verlag, New York, NY, 2004.

[10] T. P. Fries and H. G. Matthies. Classification and overview of meshfree methods. Technical Report Informatikbericht-Nr. 2003-03, Institute of Scientific Computing, Technical University Braunschweig, Braunschweig, Germany, 2004.

[11] N. Sukumar. Construction of polygonal interpolants: A maximum entropy approach. *International Journal for Numerical Methods in Engineering*, 61(12):2159–2181, 2004.

[12] M. Arroyo and M. Ortiz. Local maximum-entropy approximation schemes: a seamless bridge between finite elements and meshfree methods. *International Journal for Numerical Methods in Engineering*, 65(13):2167–2202, 2006.

[13] N. Sukumar. Maximum entropy approximation. *AIP Conference Proceedings*, 803(1):337–344, 2005.

[14] N. Sukumar and R. W. Wright. Overview and construction of meshfree basis functions: From moving least squares to entropy approximants. *International Journal for Numerical Methods in Engineering*, 70(2):181–205, 2007.

[15] N. Sukumar and R. J-B Wets. Deriving the continuity of maximum-entropy basis functions via variational analysis. *SIAM Journal of Optimization*, 18(3):914–925, 2007.

[16] N. Agmon, Y. Alhassid, and R. D. Levine. An algorithm for finding the distribution of maximal entropy. *Journal of Computational Physics*, 30:250–258, 1979.

[17] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[18] R. L. Burden and J. D. Faires. *Numerical Analysis*. Thomson/Brooks/Cole, Belmont, CA, eight edition, 2004.

[19] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization methods. *Mathematical Programming*, 45:503–528, 1989. Available at http://www.netlib.org/opt/lbfgs_um.shar.