User Manual for the BRI class in C++

CHIARA FUDA, Università della Svizzera italiana, Switzerland
KAI HORMANN, Università della Svizzera italiana, Switzerland

This user guide explains how to use the functions of the BRI C++ class template for barycentric rational interpolation, related to the article "Algorithm xxx: A C++ class for robust linear barycentric rational interpolation".

1 INSTALLATION

To use this class, include the header file BRI.h with the `#include` directive in all .cpp files that require it and place them in the same folder. Since the BRI class is using templates, the user can decide which types of input and output variables to use, and we denote them respectively by `inT` and `outT` in the following. Other than the standard data types available in C++ , these can also be defined in arbitrary precision, because the BRI class is compatible with the publicly available MPFR library using the MPFR C++ interface.

2 THE BRI CLASS

The BRI class contains all variables and functions related to a barycentric rational interpolant. Even though the variables are private, they still need to be defined by the user via constructors and can also be modified via class member functions. For this reason, we first present the parameters defined within the class and then its public functions.

2.1 Variables

`int n`

number of interpolation nodes minus 1

`int d`

integer parameter related to Floater–Hormann rational interpolation

`vector<inT>& Xn`

vector of dimension $n + 1$ containing the interpolation nodes

`vector<inT>& Yn`

vector of dimension $n + 1$ containing the data associated with the corresponding node

`vector<outT>& Wn`

vector of dimension $n + 1$ containing the barycentric weights associated with X_n and d

2.2 Constructors

`BRI(const vector<inT>& Xn, const vector<inT>& Yn, int d)`

takes as input both vectors X_n and Y_n by reference and the integer d by value

`BRI(string nodes, const vector<inT>& Yn, int d)`

fills the vector X_n with the content of the file `nodes`, takes the vector Y_n by reference and the integer d by value

Authors' addresses: Chiara Fuda, Università della Svizzera italiana, Via la Santa 1, Lugano, Switzerland, 6962, chiara.fuda@usi.ch; Kai Hormann, Università della Svizzera italiana, Via la Santa 1, Lugano, Switzerland, 6962, kai.hormann@usi.ch.

`BRI(Nodes type, inT a, inT b, int n, const vector<inT>& Yn, int d)`

generates the vector X_n by knowing the type of nodes (UNIFORM, CHEBYSHEV, CHEBYSHEV_EXTENDED) and the endpoints of the interval $[a, b]$ in which they are defined, takes the vector Y_n by reference and the integers n and d by value, as well as a and b

`BRI(const vector<inT>& Xn, string data, int d)`

takes X_n by reference and the integer d by value and fills the vector Y_n with the content of the file `data`

`BRI(string nodes, string data, int d)`

fills both vectors X_n and Y_n with the content of the files `nodes` and `data`, respectively and takes the integer d by value

`BRI(Nodes type, inT a, inT b, int n, string data, int d)`

generates the vector X_n by knowing the type of nodes (UNIFORM, CHEBYSHEV, CHEBYSHEV_EXTENDED) and the endpoints of the interval $[a, b]$ in which they are defined, fills the vector Y_n with the content of the file `data`, and takes the integers n and d by value, as well as a and b

`const BRI(vector<inT>& Xn, inT(*f)(inT x), int d)`

takes X_n by reference and the integer d by value and generates the vector $Y_n = f(X_n)$ using the pointer to an external function f

`BRI(string nodes, inT(*f)(inT x), int d)`

fills the vector X_n with the content of the file `nodes`, generates the vector $Y_n = f(X_n)$ using the pointer to an external function f, and takes the integer d by value

`BRI(Nodes type, inT a, inT b, int n, inT(*f)(inT x), int d)`

generates the vector X_n by knowing the type of nodes (UNIFORM, CHEBYSHEV, CHEBYSHEV_EXTENDED) and the endpoints of the interval $[a, b]$ in which they are defined, generates the vector $Y_n = f(X_n)$ using the pointer to an external function f, and takes the integers n and d by value, as well as a and b

2.3 Functions to modify the input

`void set_nodes(const vector<inT>& Xn)`

changes the nodes taking the new vector X_n by reference

`void set_nodes(string nodes)`

changes the nodes filling the new vector X_n with the content of the file `nodes`

`void set_nodes(Nodes type, inT a, inT b, int n)`

changes the nodes generating the new vector X_n by knowing the type of nodes (UNIFORM, CHEBYSHEV, CHEBYSHEV_EXTENDED), the endpoints of the interval $[a, b]$ in which they are defined, and the integer n, which are passed by value

`void set_nodes(int j, inT xj)`

changes only the j-th entry of the vector X_n with the value x_j

`void set_data(const vector<inT>& Yn)`

changes the data taking the new vector Y_n by reference

`void set_data(string data)`

changes the data filling the new vector Y_n with the content of the file `data`

`void set_data(inT(*f)(inT x))`

changes the data generating the new vector $Y_n = f(X_n)$ having the pointer to the external function f

`void set_data(int j, inT yj)`

changes only the j-th entry of the vector Y_n with the value y_j

`void set_degree(int d)`

changes the integer d taking the new one by value

`void add_point(inT xj, inT yj)`

adds x_j and y_j in the vectors X_n and Y_n respectively

`void remove_point(int j)`

removes the j-th entry of the vectors X_n and Y_n

2.4 Functions to get the input and the weights

`const vector<inT>& get_nodes()`

returns X_n

`const inT& get_nodes(int j)`

returns the j-th entry of the vector X_n

`const vector<inT>& get_data()`

returns Y_n

`const inT& get_data(int j)`

returns the j-th entry of the vector Y_n

`const vector<outT>& get_weights()`

returns W_n

`const outT& get_weight(int j)`

returns the jth entry of the vector W_n

`const vector<outT>& get_weights(int& Cw)`

returns W_n and the rescaling factor of the weights C_w.

2.5 Control flags

`void guard_on()`
`void guard_off()`

the guard flag can be turned on and off; if it is set, then the weights, the evaluation of the barycentric rational interpolant, and the stability-related functions are computed in guarded mode

`void stability_on()`
`void stability_off()`

the stability flag can be turned on and off; if it is set, then the code evaluates the barycentric rational interpolant using the numerically most stable algorithm

`void efficiency_on()`
`void efficiency_off()`

the efficiency flag can be turned on and off; if it is set and the evaluation of the barycentric rational interpolant is done with the first barycentric form, then the most efficient algorithm is used

2.6 Evaluation of the barycentric rational interpolant

Hereafter, all the functions that take an evaluation point x as input can be also called with a vector of evaluation points, returning a vector of values in this case. Moreover, apart from the NUMERATOR function, all the others take also the parameter A of type `Algo` as input. By default, it is set to SMART, so that, if omitted, the function autonomously decides which algorithm to use. Otherwise, it can take one value among {FIRST_DEF, FIRST_EFF, SECOND} to use the standard implementation of the first barycentric form, the efficient variant, or the second barycentric form, respectively.

`outT numerator(inT x)`
`vector<outT> numerator(vector<inT>& x)`

computes the numerator N at the evaluation point x

`outT numerator(inT x, int& CN)`
`vector<outT> numerator(vector<inT>& x, vector<int>& CN)`

computes the numerator N at the evaluation point x keeping track of the rescaling factor C_N

`outT denominator(inT x, Algo A = SMART)`
`vector<outT> denominator(vector<inT>& x, Algo A = SMART)`

computes the denominator D at the evaluation point x

`outT denominator(inT x, int& CD, Algo A = SMART)`
`vector<outT> denominator(vector<inT>& x, vector<int>& CD, Algo A = SMART)`

computes the denominator D at the evaluation point x keeping track of the rescaling factor C_D

`outT eval(inT x, Algo A = SMART)`
`vector<outT> eval(vector<inT>& x, Algo A = SMART)`

evaluates the interpolant r at the evaluation point x

`outT eval(int j, inT h, Algo A = SMART)`

evaluates the interpolant r at the evaluation point $x_j + h$

2.7 Stability-related functions

`outT cond(inT x)`
`vector<outT> cond(vector<inT>& x)`

computes the condition number $\kappa(x)$ at the evaluation point x

`outT leb(inT x)`
`vector<outT> leb(vector<inT>& x)`

computes the Lebesgue function $\Lambda_n(x)$ at the evaluation point x

`outT gamma(inT x)`
`vector<outT> gamma(vector<inT>& x)`

computes the function $\Gamma_d(x)$ at the evaluation point x

`outT cond()`

computes the value $\max_{x\in[x_0,x_n]} \kappa(x)$

`outT leb()`

computes the value $\max_{x\in[x_0,x_n]} \Lambda_n(x)$

`outT gamma()`

computes the value $\max_{x\in[x_0,x_n]} \Gamma_d(x)$

3 EXAMPLES

In this section, we provide several examples that showcase how to use the BRI class in practice.

3.1 Computation of the weights in guarded mode

We consider $n = 9$, $d = 3$, and $n + 1$ Chebyshev interpolation nodes $x_i \in [0, 10^{-12}]$ with associated data $y_i = 1$, for $i = 0, \ldots, n$. We define all variables in input and output as `float` and we compute the weights in guarded mode.

```
#include "BRI.h"

using namespace std;

int main(){
    int n = 9;
    int d = 3;
    vector<float> yn(n+1,1);

    BRI<float,float> r(CHEBYSHEV,0,1e-12,n,yn,d);
    r.guard_on();
    int C;
    vector<float> w = r.get_weights(C);

    cout.precision(20);
    cout << "Cw = " << C << endl;

    for (int i=0; i<=n; i++)
        cout << "w[" << i << "] = " << w[i] << endl;
}
```

```
Cw = -127
w[0] = -3.2477385997772216797
w[1] = 6.8478851318359375
w[2] = -5.8251581192016601562
w[3] = 3.5328421592712402344
w[4] = -2.4203362464904785156
w[5] = 2.4203360080718994141
w[6] = -3.53284454345703125
w[7] = 5.8251657485961914062
w[8] = -6.8478937149047851562
w[9] = 3.2477431297302246094
```

3.2 Evaluation of the interpolant

We consider $n = 9$, $n + 1$ Chebyshev interpolation nodes in $[0, 10^{-12}]$ and $d = 2$. We then define $y_i = f(x_i)$ for $i = 0, 1, \ldots, n$, where

$$f(x) = \tfrac{3}{4}e^{-\frac{(9x-2)^2}{4}} + \tfrac{3}{4}e^{-\frac{(9x+1)^2}{49}} + \tfrac{1}{2}e^{-\frac{(9x-7)^2}{4}} + \tfrac{1}{5}e^{-(9x-4)^2}, \tag{1}$$

and we output $r(x)$ using both the first and the second barycentric formula, both in non-guarded and guarded mode, with $x = 10^{-12}/2$. We use the functions NUMERATOR and DENOMINATOR to see the values of N, D_s, and D_f with their rescaling factors. All variables in input and output are set as `float`.

```
#include "BRI.h"

using namespace std;

float Franke(float x) {
    return exp(-(9*x-2)*(9*x-2)/4)*3/4
             +exp(-(9*x+1)*(9*x+1)/49)*3/4
             +exp(-(9*x-7)*(9*x-7)/4)/2
             +exp(-(9*x-4)*(9*x-4))/5;
}

int main(){
    int n = 9;
    int d = 2;

    BRI<float,float> r(CHEBYSHEV,0,1e-12,n,Franke,d);
    float x = 1e-12/2;
    //r.guard_on();

    int C1;
    int C2;
    int C3;

    float f_def = r.eval(x,FIRST_DEF);
    float s = r.eval(x,SECOND);
    float N = r.numerator(x,C1);
    float Ds = r.denominator(x,C2,SECOND);
    float Df = r.denominator(x,C3,FIRST_DEF);

    cout.precision(10);
    cout << "FIRST FORM - DEF: r(x) = " << f_def << endl;
    cout << "SECOND FORM: r(x) = " << s << endl;
    cout << "N = " << N << " and C1 = " << C1 << endl;
    cout << "Ds = " << Ds << " and C2 = " << C2 << endl;
    cout << "Df = " << Df << " and C3 = " << C3 << endl;
}
```

```
GUARD FLAG TURNED OFF

FIRST FORM - DEF: r(x) = -nan
SECOND FORM: r(x) = -nan
N = -nan and C1 = 0
Ds = -nan and C2 = 0
Df = inf and C3 = 0

GUARD FLAG TURNED ON

FIRST FORM - DEF: r(x) = 1.010761023
SECOND FORM: r(x) = 1.010760903
N = 5.134361744 and C1 = -44
Ds = 10.15939903 and C2 = -43
Df = 40.63759232 and C3 = -125
```

3.3 Evaluation of the interpolant close to a node

We consider $n = 9$, $d = 2$, and $n + 1$ equidistant interpolation nodes $x_i \in [1, 2]$ with associated data $y_i = f(x_i)$ for $i = 0, 1, \ldots, n$ and $f(x) = x$. We evaluate the interpolant r very close to the first node $x_0 = 1$ at $x = x_0 + h$ for $h = 10^{-20}$, and we set all variables in input and output as `double`.

```cpp
#include "BRI.h"

using namespace std;

int main(){
    int n = 9;
    int d = 2;

    double a = 1;
    double b = 2;
    vector<double> xn = uniform(a,b,n);
    double h = 1e-20;
    double x = xn[0] + h;

    BRI<> r(xn,xn,d);

    double r1 = r.eval(x);
    double r2 = r.eval(0,h);

    cout.precision(20);
    cout << "eval(x) outputs:" << endl;
    cout << "r(x) = " << r1 << endl;
    cout << "eval(0,h) outputs:" << endl;
    cout << "r(h) = " << r2 << endl;
}
```

```
eval(x) outputs:
r(x) = 1
eval(0,h) outputs:
r(h) =  9.999999999999949376e-21
```

3.4 Evaluation of the stability-related functions

We consider $n = 9$, $n + 1$ equidistant interpolation nodes $x_i \in [0, 1]$ with associated data $y_i = f(x_i)$ for $i = 0, \ldots, n$ and $f(x) = 1/(1 + 25x^2)$ and $d = 3$. We create a new instance of the BRI class that takes double input and returns multiple precision (1024 bits) output using the MPFR library, and we ask for the maximum of the functions Λ_n, Γ_d, and κ.

```cpp
#include "mpreal.h"
#include "BRI.h"

using namespace std;
using mpfr::mpreal;

double Runge(double x){
    return 1/(1+25*x*x);
}

int main(){
    int my_mpreal_precision = 1024;
    mpreal::set_default_prec(my_mpreal_precision);

    int n = 9;
    int d = 3;
    BRI<double,mpreal> r(UNIFORM,0,1,n,Runge,d);

    cout.precision(20);
    cout << "Maximum of the Lebesgue function:" << endl;
    cout << r.leb() << endl;
    cout << "Maximum of the function Gamma:" << endl;
    cout << r.gamma() << endl;
    cout << "Maximum of the condition number:" << endl;
    cout << r.cond() << endl;
}
```

```
n = 9

Maximum of the Lebesgue function:
3.5886287189761606401
Maximum of the function Gamma:
1.0318045847764588507
Maximum of the condition number:
11.609466977862612706

n = 19

Maximum of the Lebesgue function:
4.6127100859322925745
Maximum of the function Gamma:
1.0322814978345268598
Maximum of the condition number:
13.210805972626199269

n = 39

Maximum of the Lebesgue function:
5.5370777898252804523
Maximum of the function Gamma:
1.0323229058478393483
Maximum of the condition number:
14.979125760718810308
```