

paxx:  
Computing piece-wise analytic approximations to  
probability distributions and potentials of mean force

*User Documentation*

Ramses van Zon\*

*Chemical Physics Theory Group, Department of Chemistry, University of Toronto,*

*80 St. George Street, Toronto, Ontario M5S 3H6, Canada*

December 14, 2009

**Abstract**

This document describes how to use the functions defined in the C++ header file *paxx.h* and implemented in the file *paxx.cc*. These functions can compute piece-wise analytic approximations to probability distributions and potentials of mean force based on sample data.

---

\*rzon@chem.utoronto.ca

## Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>3</b>
<b>2</b>	<b>HEADER FILE AND LINKAGE</b>	<b>4</b>
<b>3</b>	<b>EXAMPLE</b>	<b>4</b>
<b>4</b>	<b>DEFINITION OF class pa::Fit</b>	<b>5</b>
<b>5</b>	<b>FUNCTIONS</b>	<b>6</b>
5.1	pa::bias . . . . .	6
5.2	pa::jbias . . . . .	6
5.3	pa::boot . . . . .	7
5.4	pa::jack . . . . .	7
5.5	pa::jackboot . . . . .	8
5.6	pa::newreadbinfile . . . . .	9
<b>6</b>	<b>MEMBER FUNCTIONS OF THE CLASS pa::Fit</b>	<b>9</b>
6.1	pa::Fit::fit . . . . .	9
6.2	pa::Fit::cd . . . . .	10
6.3	pa::Fit::pd . . . . .	10
6.4	pa::Fit::write . . . . .	11
	<b>BACKGROUND REFERENCES</b>	<b>11</b>

# 1 INTRODUCTION

`paxx` is a set of functions to implement a method of obtaining smooth analytical estimates of probability densities, radial distribution functions and potentials of mean force from sampled data in a statistically controlled fashion. `paxx` is the `c++` version of the `c` package `pa`. The fitting method only uses direct samples of data (distance samples in the case of radial distribution functions). The result is a piecewise analytic function, such that the probability distribution is described in different domains by different truncated fourier series.

Briefly stated, the approach consists of the following:

1. Gather/read in sample data (using for example `pa::newreadbinfile`).
2. In the case of radial distribution functions, perform a biased resampling of the data (using `pa::bias` or variations). This step can often be omitted for probability distribution functions.
3. Sort the data (using for example `std::sort`).
4. Feed the sorted data to the `fit` member function of instances of the `Fit` class; this creates a division of the range of the samples in different intervals, a truncated fourier series in each interval, and, optionally, a quadratic patch at the division points between the intervals. All this data is stored in the class, and need not be accessed by the user.
5. The results of `fit` can be used to evaluate the probability distribution function or cumulative at any point within the range of the data.

In addition, it is possible to tune the following parameters:

- The upper limit on the number of fourier modes in each interval. A good default value is 14.
- The upper limit on the number of intervals. A good default value is 21.
- The acceptable 'quality factor', lying between 0 and 1. A good default value is 0.6.

The piecewise Fourier approach can be applied to any density of a single random variable. The method outlined here avoids the use of histograms, which require the specification of a physical parameter (bin size) and tend to give noisy results. The technique used is an extension of an existing method[1], which is typically inaccurate for radial distribution functions and potentials of mean force due to a non-uniform Jacobian factor. In addition, the standard method often requires a large number of Fourier modes to represent radial distribution functions, which tends to lead to oscillatory fits. It is shown that the issues of poor sampling due to a Jacobian

factor can be resolved using a biased resampling scheme, while the requirement of a large number of Fourier modes is mitigated through an automated piecewise construction approach.

The theoretical underpinning of this method can be found in Ref. [2], which also contains example applications.

## 2 HEADER FILE AND LINKAGE

All routines are defined in the file *paxx.h*. The implementation is found in *paxx.cc*. Note that all functions, as well as the `Fit` class are encapsulated in the namespace `pa`.

## 3 EXAMPLE

```

/* testpafit.c
   Example of the piece-wise analytic fit to a probability distribution.
   Ramses van Zon, December 14, 2009 */
#include <iostream>
#include <algorithm>
#include "paxx.h"
const int km = 21; /* maximum number of intervals */
const int mm = 14; /* maximum number of Fourier modes per interval */
const double qm = 0.6; /* satisfactory value for Q */
int main( int argc, char **argv ) {
    int m, n;
    double q, y, y1, y2, dy, *r;
    pa::Fit fit;
    r = newreadbinfile(argv[1], y1, y2, m, n);
    std::sort(n,r);
    q = fit.fit(n,r,qm,km,mm,true);
    delete []r;
    dy = (y2-y1)/m;
    for (y = y1; y <= y2; y += dy)
        std::cout << y << " " << fit.pd(y) << " " << fit.cd(y) << std::endl;
    std::cout << "#Parameters (q=" << q << ")" << std::endl;
    fit.pawrite(stdout);
}

```

## 4 DEFINITION OF class pa::Fit

The following class is used to hold the parameters of the piecewise analytic fit.

```
class pa::Fit {
private:
    int km_;
    int mm_;
    int k;
    double *a;
    double *b;
    double *c;
    double **d;
    double *f;
    double *g;
    void dealloc();
    void realloc(int km,int mm);
public:
    Fit();
    ~Fit();
    double fit(int n,const double*r,double qm,int km,int mm,bool nodiscont);
    double pd(double r);
    double cd(double r);
    void write(FILE*file);
}; .
```

For debugging purposes, it may be useful to know the meaning of the private members, which is as follows:

- `km_` is the maximum number of intervals in the fit
- `mm_` is the maximum number of Fourier modes in each interval
- `k` is the number of intervals in the fit
- `a` is the array of limits of the intervals
- `b` is the array of patch coefficients
- `c` is the array of patch widths
- `d` are the Fourier coefficients, with `d[0]` the number of coefs
- `f` is the array of relative interval weights
- `g` is the array of cumulative interval weights
- `realloc` allocates enough memory to hold the fit with at most `km` intervals with `mm` modes.
- `dealloc` releases this memory.

## 5 FUNCTIONS

### 5.1 `pa::bias`

`pa::bias` performs a weighted resampling of data.

#### Definition

```
double pa::bias(int n,const double*r,double(*w)(double),
                double(*rnd)(void*),void*s,double*x);
```

#### Parameters

1. `n` number of data points (input)
2. `r` original data (input)
3. `w` weighing function [e.g. `double w(double r){return r*r;}`] (input)
4. `rnd` a random number generating function  
[e.g. `double rnd(void*){return double(rand())/MAX_RAND;}`]  
(input)
5. `s` pointer to the seed for the random number generator (input/output)
6. `x` resampled data (output)

#### Return value

the normalization correction factor ( $z$ )

---

### 5.2 `pa::jbias`

`pa::jbias` performs a weighted resampling of data for use with a block-jackknife estimator.

#### Definition

```
double pa::jbias(int j,int m,int n,const double*r,double(*w)(double),
                 double(*rnd)(void*),void*s,double*x);
```

---

**Parameters**

1. j what block to omit (input)
2. m total number of blocks (input)
3. n number of data points (input)
4. r original data (input)
5. w weighing function [e.g. double w(double r)return r\*r;] (input)
6. rnd a random number generating function  
[e.g. double rnd(void\*){return double(rand())/MAX\_RAND;}]  
(input)
7. s pointer to the seed for the random number generator (input/output)
8. x resampled data (output)

---

**5.3 pa::boot**

pa::boot performs a resampling of data for use with a bootstrap estimator.

**Definition**

```
void pa::boot(int n, const double*r, double(*rnd)(void*), void*s, double*x);
```

**Parameters**

1. n number of data points (input)
2. r original data (input)
3. rnd a random number generating function  
[e.g. double rnd(void\*){return double(rand())/MAX\_RAND;}]  
(input)
4. s pointer to the seed for the random number generator (input/output)
5. x resampled data (output)

---

**5.4 pa::jack**

pa::jack performs a simple block-jackknife.

**Definition**

```
void pa::jack(int j,int m,int n,const double*r,double*x);
```

**Parameters**

1. j what block to omit (input)
  2. m total number of blocks (input)
  3. n number of data points (input)
  4. r original data (input)
  5. x resampled data (output)
- 

**5.5 pa::jackboot**

pajackboot performs a resampling of data for use with a block-jackknife estimator.

**Definition**

```
void pajackboot(int j,int m,int n,const double*r,double(*rnd)(void*),void*s,double*x);
```

**Parameters**

1. j what block to omit (input)
  2. m total number of blocks (input)
  3. n number of data points (input)
  4. r original data (input)
  5. rnd a random number generating function  
[e.g. double rnd(void\*){return double(rand())/MAX\_RAND;}]  
(input)
  6. s pointer to the seed for the random number generator (input/output)
  7. x resampled data (output)
-

## 5.6 pa::newreadbinfile

pa::readbinfile reads a binary data file of a specific format into an array, for which it allocates the memory using malloc. This routine is only provided for convenience, its use is not required. All numbers stored in the file should be doubles. The first three numbers should contain the smallest number in the file (y1), the largest number in the file (y2), and the number of data points to be plotted (m). These three numbers are only used for informational purposes. The remaining numbers in the file are data samples, which read in and stored an array that readbinfile allocates. Once the data is analyzed, the memory taken up by this array may be released using the standard "free" c function. Note that this routine does not check for lsb/msb order.

### Definition

```
double*pa::newreadbinfile(const char*s,double*y1,double*y2,int*m,int*n);
```

### Parameters

1. s string with the file name of the data file (input)
2. y1 pointer to the double to contain the first number in the file (output)
3. y2 pointer to the double to contain the second number in the file (output)
4. m pointer to the integer to contain the third number in the file (output)
5. n pointer to the integer to be set to the number of points (output)

### Return value

a pointer to the allocated array of n doubles containing the data

## 6 MEMBER FUNCTIONS OF THE CLASS pa::Fit

### 6.1 pa::Fit::fit

pafit computes the parameters of the piecewise analytic fit to a probability distribution from data.

### Definition

```
double pafit(int n,const double*r,double qm,int km,int mm,bool nodiscont)
```

**Parameters**

1. `n` number of data points (input)
2. `r` array of sorted data points (input)
3. `qm` maximally allowed value for  $Q$  (input)
4. `km` maximum number of intervals (input)
5. `mm` maximum number of Fourier modes in each interval (input)
6. `nodiscont`: set this to true if no discontinuities are to be present in the result.

**Return value**

the goodness-of-fit parameter  $q$

---

**6.2** `pa::Fit::cd`

`pa::Fit::cd` evaluates the piecewise analytic cumulative distribution.

**Definition**

```
double pa::Fit::cd(double r);
```

**Parameters**

1. `r` the point at which to compute the cumulative distribution (input)

**Return value**

The cumulative distribution at  $r$

---

**6.3** `pa::Fit::pd`

`pa::Fit::pd` evaluates the piecewise analytic probability distribution at a point  $r$ . If the data used to compute the cumulative distribution, then the result should be divided by the bias at the point  $r$  and multiplied by the normalization correction factor. evaluates the piecewise analytic cumulative distribution.

**Definition**

```
double pa::Fit::pd(double r);
```

**Parameters**

1. r the point at which to compute the probability distribution function (input)

**Return value**

The probability distribution function at r

---

**6.4 pa::Fit::write**

pa::Fit::write writes the fit parameters found by pafit to a file.

**Definition**

```
void pa::Fit::write(FILE*file);
```

**Parameters**

1. file a c FILE pointer
- 

**BACKGROUND REFERENCES**

- [1] B. A. Berg and R.C. Harris, *Comp. Phys. Comm.* **179**, 443 (2008).
- [2] Ramses van Zon and Jeremy Schofield, *Constructing smooth potentials of mean force, radial distribution functions and probability densities from sampled data* arxiv.0912.0465 [cond-mat.stat-mech] (2009).
- [3] The comments in paxx.cc.