

pa:  
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 5, 2009

**Abstract**

This document describes how to use the functions defined in the C header file *pa.h* and implemented in the files *pa1.c*, *pa2.c* and *pautil.c*. 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>LAYOUT OF struct Pafit</b>	<b>5</b>
<b>5</b>	<b>FUNCTIONS</b>	<b>5</b>
5.1	pabias . . . . .	5
5.2	pajbias . . . . .	6
5.3	paboot . . . . .	7
5.4	pajack . . . . .	7
5.5	pajackboot . . . . .	8
5.6	paalloc . . . . .	8
5.7	pafree . . . . .	9
5.8	pafit . . . . .	9
5.9	pacd . . . . .	10
5.10	papd . . . . .	10
5.11	pawrite . . . . .	11
5.12	pasort . . . . .	11
5.13	readbinfile . . . . .	12
	<b>BACKGROUND REFERENCES</b>	<b>12</b>

# 1 INTRODUCTION

`pa` 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. This 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 `readbinfile`).
2. In the case of radial distribution functions, perform a biased resampling of the data (using `pabias` or variations). This step can often be omitted for probability distribution functions.
3. Sort the data (using for example `pasort`).
4. Feed the sorted data to the `pafit`; this function returns 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 a struct, and need no be accessed by the user.
5. The results of `pafit` 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 *pa.h*. To avoid having to link the routines into every executable, the implementation is spread out over three files:

1. *pa1.c* contains the implementation of the functions *pajack*, *paalloc*, *pafree*, *pafit*, *papd*, *pacd*, *pawrite*. These are the functions which do not require a random number generator.
2. *pa2.c* contains the implementation of the functions *paboot*, *pajackboot*, *pabias* and *pajbias*. These function require a random number generator, which has to be supplied by the user (see below).
3. *pautil.c* contains implementations of a sorting routine *pasort* and a routine to read binary data files, *readbinfile*. These routines are optional and the user can link use their own.

## 3 EXAMPLE

```

/* testpafit.c
   Example of the piece-wise analytic fit to a probability distribution.
   Ramses van Zon, December 5, 2009 */
#include <stdio.h>
#include <stdlib.h>
#include "pa.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;
    struct Pafit *fit;
    r = readbinfile(argv[1], &y1, &y2, &m, &n);
    fit = paalloc(km,mm);
    pasort(n,r);
    q = pafit(n,r,qm,km,mm,1,fit);
    free(r);
    dy = (y2-y1)/m;
    for (y = y1; y <= y2; y += dy)
        printf("%f %f %f/n", y, papd(y,fit), pacd(y,fit));
    printf("#Parameters (q=%f):/n", q);
    pawrite(stdout,fit);
    pafree(fit);
}

```

## 4 LAYOUT OF struct Pafit

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

```
struct Pafit {
    int k;
    double *a;
    double *b;
    double *c;
    double **d;
    double *f;
    double *g;
};
```

Here:

- 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

## 5 FUNCTIONS

### 5.1 pabias

pabias performs a weighted resampling of data. The implementation assumes a global random number generator function called `rnd(...)` exists taking a pointer and returning a double between 0 and 1.

#### Definition

```
double pabias(int n, const double*r, double>(*w)(double), 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. s pointer to the seed for the random number generator (input/output)
5. x resampled data (output)

**Return value**

the normalization correction factor (z)

---

**5.2 pajbias**

pajbias performs a weighted resampling of data for use with a block-jackknife estimator. The implementation assumes a global random number generator function called rnd(..) exists taking a pointer and returning a double between 0 and 1.

**Definition**

```
double pajbias(int j,int m,int n,const double*r,double(*w)(double),  
               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. s pointer to the seed for the random number generator (input/output)
  7. x resampled data (output)
-

### 5.3 paboot

paboot performs a resampling of data for use with a bootstrap estimator. The implementation assumes a global random number generator function called `rnd(. .)` exists taking a pointer and returning a double between 0 and 1.

#### Definition

```
void paboot(int n,const double*r,void*s,double*x);
```

#### Parameters

1. n number of data points (input)
  2. r original data (input)
  3. s pointer to the seed for the random number generator (input/output)
  4. x resampled data (output)
- 

### 5.4 pajack

pajack performs a simple block-jackknife.

#### Definition

```
void pajack(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 pajackboot

pajackboot performs a resampling of data for use with a block-jackknife estimator. The implementation assumes a global random number generator function called rnd(..) exists taking a pointer and returning a double between 0 and 1.

### Definition

```
void pajackboot(int j,int m,int n,const double*r,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. s pointer to the seed for the random number generator (input/output)
6. x resampled data (output)

---

## 5.6 paalloc

paalloc allocates memory for the arrays used by pafit.

### Definition

```
struct Pafit* paalloc(int km,int mm);
```

### Parameters

1. km maximum number of intervals (input)
2. mm maximum number of Fourier modes in each interval (input)

### Return value

A pointer to an allocated struct Pafit structure. For the layout of this structure see section 4.

---

## 5.7 pafree

pafree releases the memory allocated by paalloc.

### Definition

```
void pafree(struct Pafit *pa);
```

### Parameters

1. pa the pointer to a struct Pafit to be freed (input).
- 

## 5.8 pafit

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,int nodiscont,  
            struct Pafit*pa);
```

### 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 1 if no discontinuities are to be present in the result
7. pa pointer to a struct Pafit to contain the result.

### Return value

the goodness-of-fit parameter q

---

## 5.9 pacd

pacd evaluates the piecewise analytic cumulative distribution.

### Definition

```
double pacd(double r,struct Pafit*pa);
```

### Parameters

1. r the point at which to compute the cumulative distribution (input)
2. pa pointer to parameters of the fit

### Return value

The cumulative distribution at r

---

## 5.10 papd

papd 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 papd(double r,struct Pafit*pa);
```

### Parameters

1. r the point at which to compute the probability distribution function (input)
2. pa pointer to parameters of the fit

### Return value

The probability distribution function at r

---

## 5.11 pawrite

pawrite writes the fit parameters found by pafit to a file.

### Definition

```
void pawrite(FILE*file,struct Pafit*pa);
```

### Parameters

1. file a c FILE pointer
  2. pa pointer to parameters of the fit
- 

## 5.12 pasort

pasort is a sorting routine for an array of doubles. It performs a recursive quicksort. The routine pasort is only provided for convenience.

### Definition

```
void pasort(int n,double*x);
```

### Parameters

1. n number of data points (input)
  2. x input: pointer to unsorted data; output: sorted data
- 

## 5.13 readbinfile

readbinfile reads a binary data file of a specific format into an array, for which it allocate 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*readbinfile(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

---

**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 `pa1.c` and `pa2.c`.