ChiExp Documentation

Release 0.1.0

Mattia Bruno, Rainer Sommer

Aug 30, 2022

A package to compute the expectation value of the χ^2 defined from arbitrary weight matrices W

$$\chi^2 = \sum_{i,j} [y_i - f(x_i,a)] W_{ij} [y_j - f(x_j,a)]$$

The implementation is based on the results of Ref.[1] and we summarize below the main equation

$$\langle \chi^2 \rangle = \mathrm{tr}\left[C_W W^{1/2} (1-P) W^{1/2} \right], \quad C_W = W^{1/2} C W^{1/2}$$

with C being the covariance matrix and P a projector, defined from the function f and its derivatives computed at the minimum of the χ^2 (for more details read Ref.[1]).

Autocorrelations can be taken into account in a straight-forward manner by replacing

$$C_{ij} = \frac{1}{N} \sum_{t=-\infty}^{\infty} \Gamma_{ij}(t)\,,$$

with N the number of configurations and Γ the autocorrelation function.

The package contains libraries for both Matlab and Python.

1

M. Bruno and R. Sommer title

CHAPTER

ONE

MATLAB DOCUMENTATION

The matlab package contains three functions:

derfit computes the derivatives of the parameters w.r.t. the input observable Y at the minimum

chiexp computes the expected χ^2

qfit computes the quality of fit

1.1 derfit

At the minimum of the χ^2, the parameters depend on the input observables Y. This function computes their derivatives w.r.t. Y:

```
der = derfit(X,Y,W,p,'field',value);
```

Parameters

- **X, Y** - arrays with N entries; Y is assumed to contain the central values of the observable
- **W** - weight matrix, N-by-N, used in the fit
- **p** - array with the values of the parameters at the minimum
- **field** - a string whose admitted values as `'f'` and `'df'`
- **value** - the value corresponding to one of the two strings above
  - **df** - the gradient of the function f. An array of functions is expected `df = @(x,p) [df/dp1 df/dp2 ...]`
  - **f** - fitted function. It must be passed in the form `f=@(x,p) f(x,p)`; if passed together with *df* it is used to check numerically the provided gradient; if passed without *df* it is used to numerically compute the gradient

Additional arguments

- **c** *(optional)*: array of length N used as additional argument for *f* and *df*, which in this case must obey the syntax `f = @(x,p,c) f(x,p,c)` (same for *df*); it can be useful for global fits

Returns

- **der**: a N-by-NA matrix, with NA the number of parameters; contains the derivatives of the parameters `der(i,a) = dp(a)/dy(i)`

Examples:

```
% derfit can be used in two ways:
% either by passing the function (numerical gradient)
func = @(x,p) p(1) + p(2)*x;
der = derfit(X,Y,W,p,'f',func);

% or by passing the gradient (analytical)
grad = @(x,p) [1, x];
der = derfit(X,Y,W,p,'df',grad);

% if both are passed, the function is used to check the
% gradient
der = derfit(X,Y,W,p,'f',func,'df',grad);

% additional argumemts
der = derfit(X,Y,W,p,'f',func,'df',grad,'c',c);
```

1.2 chiexp

This function computes the expected χ^2. The covariance matrix is automatically estimated from the fluctuations of Y, but the user can also provide it independently:

```
[ce,dce,nu,covest] = chiexp(X,Y,W,p,cov,'field',value)
```

Parameters

- **X, Y** - arrays with N entries
- **W** - weight matrix. It can be an array of dimensions N or a matrix of dimension NxN. In the first case a diagonal weight matrix is automatically created
- **p** - array with the values of the parameters at the minimum
- **cov** - if cov is a N-by-N matrix, the programs takes it as the covariance matrix, assuming the user has previously computed it; if cov is a M-by-N matrix, the program assumes that it contains the values of the N observables Y on M different configurations.
- **field** - a string whose admitted values as `'f'` and `'df'`
- **value** - the value corresponding to one of the two strings above
  - **f** - fitted function. It must be passed in the form `f = @(x,p) f(x,p)`; this argument is ignored if *df* is passed
  - **df**: the gradient of the function f. An array of functions is expected `df = @(x,p) [df/dp1 df/dp2 ...]`. If empty, the gradient is computed numerically and *f* must be passed

Additional parameters can be passed using the syntax:

```
[ce,dce] = chiexp(X,Y,W,p,cov,'df',df,'field1',value1,'field2',value2...)
% or
[ce,dce] = chiexp(X,Y,W,p,cov,'f',f,'field1',value1,'field2',value2...)
```

Additional arguments

- **c** *(optional)*: array of length N used as additional argument for *f* and *df*, which in this case must obey the syntax `f = @(x,p,c) f(x,p,c)` (same for *df*); it can be useful for global fits

- **Nrep** *(optional)*: array with the number of configurations belonging to each replica, e.g. `[N1 N2 N3 ...]`. If not given, the number of replica is assumed to be 1.
- **Stau** *(optional)*: value of the parameter Stau. Default is 1.5.
- **plot** *(optional)*: flag to produce a plot of the expected χ^2 as a function of the window. Default is `'off'`. Accepted values are `'on'` and `'off'`.

Returns

- **ce, dce** - the values of $\langle\chi^2\rangle$ and its error
- **nu**: the matrix $\left[C^{1/2}W^{1/2}(1-P)W^{1/2}C^{1/2}\right]$
- **covest**: the estimated covariance matrix using the window obtained from the expected χ^2 (which may not be adeguate for general purposes)

Examples:

```
% chiexp can be used in two ways:
% either by passing the function (numerical gradient)
func = @(x,p) p(1) + p(2)*x;
[ce,dce] = chiexp(X,Y,W,p,cov,'f',func);

% or by passing the gradient (analytical)
grad = @(x,p) [1, x];
[ce,dce] = chiexp(X,Y,W,p,cov,'df',grad);

% if cov is M-by-N and has 2 replica
Nrep=[N1, N2]; % M=N1+N2
[ce,dce] = chiexp(X,Y,W,p,cov,'df',grad,'Nrep',Nrep,'Stau',2.5);

% full output
[ce,dce,nu,covest] = chiexp(X,Y,W,p,cov,'df',grad);

% for W^-1 = diag(C)
[ce,dce,nu,covest,ce2,dce2] = chiexp(X,Y,W,f,p)
```

Note: If W is an array of length N corresponding to the diagonal entries of the covariance matrix, the expected χ^2 can be obtained with a second formula. To have both results the user must request two additional output arguments, that will contain $\langle\chi^2\rangle = N - \mathrm{tr}\left[CW^{1/2}PW^{1/2}\right]$ and its error

1.3 qfit

The quality of fit and its error are estimated from a Monte Carlo integration.:

```
[Q,dQ] = qfit(nmc,nu,c2,plot)
```

Input arguments:

- *nmc*: the size of the Monte Carlo chain
- *nu*: the matrix returned from the function *chiexp*
- *c2*: the value of the χ^2 obtained from the fitting procedure

- ***plot*** **(*optional*): is passed a histogram with normalized distribution** probability obtained from the Monte Carlo process is plotted

Returns:

- *Q,dQ*: the quality of fit and its error

CHAPTER

TWO

PYTHON DOCUMENTATION

The package *chiexp* can be loaded in python using the standard import command:

```python
import sys
sys.path.append('/path/to/chiexp/directory/lib/python')
from chiexp import chisquare
```

2.1 The chisquare class

class chiexp.**chisquare**(*x*, *y*, *W*, *f*, *df*, *v='x'*)

A class with the relevant functionalities to compute the expected chi square.

Parameters

- **x** (*array*) – array with the values of the x-axis; 2-D arrays are accepted and the second dimension is intepreted as internal kinematic index
- **y** (*array*) – array with the values of the y-axis, of same length as *x*
- **W** (*array*) – the weight matrix, N-by-N, or its diagonal of length N
- **f** (*function*) – callable function or lambda function defining the fitted function; the program assumes *x* correspond to the first arguments
- **df** (*function*) – callable function or lambda function returning an array that contains the gradient of *f*, namely $\partial\phi(\{p\}, \{x_i\})/\partial p_\alpha$
- **v** (*str, optional*) – a string with the list of variables used in *f* as the kinematic coordinates. Default value corresponds to *x*, which implies that *f* must be defined using *x* as first and unique kinematic variable.

chiexp(*cov*, *Nrep=None*, *Stau=1.5*, *Wopt=None*, *Wcov=None*, *plot=False*)

Computes the expected chi square

Parameters

- **cov** (*list or array*) – the covariance matrix is assumed if the object passed has is a N-by-N 2D array; otherwise if it is a M-by-N 2D array the program assumes that it contains the fluctuations of the N observables over M configurations
- **Nrep** (*list, optional*) – number of configurations per replica, e.g. if `Nrep=[N1,N2,N3]` then `M=N1+N2+N3`
- **Stau** (*float, optional*) – parameter used in the automatic window procedure

- **Wopt** (*float, optional*) – optimal window used in the estimate of expected chi square; if passed *Stau* is ignored
- **Wcov** (*float, optional*) – window used in the estimate of all entries of the covariance matrix
- **plot** (*bool, optional*) – if set to *True* the program produces a plot with the autocorrelation function of the expected chi square

Returns

the expected chi square, its error and the estimated covariance matrix; if *Wcov* is not passed the window obtained from the autocorrelation function of the expected chi square is used for all elements; if *cov* is a N-by-N 2D array it returns a copy of it

Return type list

Note: The additional parameters *Nrep*,`Stau` and *plot* are ignored if *cov* corresponds to the covariance matrix, i.e. it is a N-by-N array. The matrix $[C^{1/2}W^{1/2}(1-P)W^{1/2}C^{1/2}]$ can be accessed by invoking the `nu` element of the `chisquare` class

chisq(*p*)

Returns the chi square from the input parameters *p*.

derfit()

Returns the derivative of the parameters w.r.t. Y

At the minimum the parameters depend on the input observables; their errors can be obtained by error propagation through the derivatives provided by this routine.

Returns the derivatives `der(i,a) = dp(a)/dy(i)`

Return type array

fit(*p0*, *min_search*)

Minimizes the chi square starting from the initial guess parameters *p0*

Parameters

- **p0** (*array*) – the initial guess values of the parameters
- **min_search** (*function*) – an external minimizer

Returns output parameters and the value of the chi square at the minimum

Return type list

pvalue(*method='eig'*, *nmc=5000*, *plot=False*)

Computes the p-value of the fit

Parameters

- **method** (*string, optional*) – string specifying the method to estimate the quality of fit. Accepted values are 'MC' for a pure Monte Carlo estimate or 'eig' (default) for the formula based on the eigenvalues of the matrix *nu*.
- **nmc** (*int, optional*) – number of Monte Carlo samples used to estimate the quality of fit. Default is 5000.
- **plot** (*bool, optional*) – if set to True plots the probabilty distribution of the expected chi square

Returns the quality of fit, the error of the quality of fit and the Monte Carlo history of the expected chi square

Return type list

Note: The class must know the value of the chi square at the minimum, which means that either *fit* or *chisq* must be called before *qfit*. If method is set to 'MC' the error of the quality of fit is based only the MC sampling.

set_pars(*p0*)
Sets the parameters to the input values *p0*

2.2 Examples of usage

First we define a new instance of the *chisquare* class; in the following example we do it for an uncorrelated fit

```
[x, y, dy] = load_your_data_set()

# if dy is the error of y we define W for an uncorrelated fit
W = [1./e**2 for e in dy]

func = lambda x, a, m: a*exp(-m*x)
dfunc = lambda x, a, m: [exp(-m*x), -a*x*exp(-m*x)]

c=chisquare(x,y,W,func,dfunc)
```

If the minimum of χ^2 is not know the user can use the `fit` method; otherwise the values of the parameters at the minimum can be passed to the class via the `chisq` method

```
>>> from scipy.optimize import minimize
>>> p0=[1.,1.] # guess values of the paramters
>>> [p, c2] = c.fit(p0, minimize)
>>> # or pass the parameters at the minimum
>>> c.chisq(p)
```

The user can also compute the error of the fitted parameters from the errors (fluctuations) of the input observables Y. To do so the derivatives dp_α/dy_i (defined at the minimum of the χ^2) are needed and if the covariance matrix is known, applying the chain rule returns the wanted errors

```
>>> der = c.derfit() # N x Na matrix
>>> print (der.T @ cov @ der) # errors of parameters
```

$\langle\chi^2\rangle$ can be computed using the method *chiexp*

```
>>> [ce, dce, _] = c.chiexp(cov)
>>> # if cov contains the fluctuations of M configs and 2 replica
>>> (M, N) = numpy.shape(cov)
>>> nr = [N1 N2] # such that M=N1+N2
>>> print M - sum(nr)
0
>>> [ce,dce,covest] = c.chiexp(cov,Nrep=nr,Stau=2.5)
```

Finally the quality of fit is estimated from a Monte-Carlo chain of length *nmc*

```
>>> nmc=10000
>>> [p, dp, h] = c.pvalue(nmc, plot=True)
```

The method *pvalue* returns the quality of fit and its error, *p* and *dp* respectively, and the Monte Carlo history of χ^2 *h*. If the *plot* flag is activated a plot is automatically generated with the distribution probability, namely a normalized histogram of *h*.

Further details can be found by typing *help chiexp* in Matlab or *help(chisquare)* in Python.

CHAPTER

THREE

REFERENCES

PYTHON MODULE INDEX