

Fit Routine

This next function fitl perform least squares fitting of the routine, thus it is the nucleus of the project.

Inside this function you can access the data and call other useful functions in respect to it.

The x,y,err,w data is can be access form a pointer named downer. For example to get the nth value of x, y, err, and the weight one would call the below statements respectively

```
(*downer->x)[n-1];
(*downer->y)[n-1];
(*downer->err)[n-1];
(*downer->w)[n-1];
```

Other useful functions can be called from this pointer these are

1. n(); returns number of points in the dataset.
2. dmeanx() returns mean of the x data
3. dmeany() returns mean of the y data
4. dminx() returns minimum x value
5. dminy() returns maximum y value
6. dmaxx() returns maximum x value
7. dmaxy() returns maximum y value
8. wxiN(i,no) returns the product of x and weight for the ith value
9. EwxN(no) returns the sum of product of the weight and x value raised to the no power for all values in the dataset.
10. EwyxN(no) returns the sum of product of the weight, y value, and x value raised to the no power for all values in the dataset.
11. Ew() returns the sum of the weights for all values in the dataset.
12. Ewy() returns the sum of product of the weight and y value for all values in the dataset.

For example to return the sum of the product of the weight and the x value squared for all values in the dataset would involve writing.

```
downer->EwxN(2);
```

The parameters of a fit and their errors values are stored in the double arrays lconst[] errconst[] respectively. Thus this function must involve setting these to their respective values. It is important also to note that any estimated values provided by the user will appear in the lconst[] array.

Power Routine

As shown below this form of fit can be linearized by a change of variables.

$$\log y = \log A + \frac{B}{2} \log x^2$$

Thus it then can be solved by the polynomial routine for 1st order.

Thus the few first statement is create a new dataset to store these values.

```
dataset tmpd(0); // creates new dataset
tmpd.x = new row<double>(downer->x->no()); // create a new row with length of the x component of
the orginal dataset
tmpd.y = new row<double>(downer->y->no()); // same but y
tmpd.err = new row<double>(downer->erry->no()); // same but error in y
```

The next step is to populate these rows will the appropriate values

```
for (int a = 0; a < downer->x->no(); a++) { // populate x,y,erry for new dataset
    (*tmpd.x)[a] = logg((*downer->x)[a] * (*downer->x)[a]); new x=log(x^2)
    (*tmpd.y)[a] = logg((*downer->y)[a]); // new y = log(y)
    (*tmpd.erry)[a] = (*downer->erry)[a] / (*downer->y)[a]; } // new err = err/y
```

```
tmpd.calw(); // calculate weights
```

Now we need to create a instance of the polynomial routine for 1st order and call it.

```
polynom tmpf(1); // create a instance of a fit polynomial of the form y=ax+b lconst[0] = a, lconst[1]
= b
useotherfit(&tmpf); // perform the fit log(y)=log(A) + (B/2)*log(x^2)
```

Now we need to convert the found fitted parameters to ones that are expected by the equ function.

```
lconst[0] = poww(10 , tmpf.lconst[1]); // determine A from fit
lconst[1] = tmpf.lconst[0]*2; // determine B from fit
errconst[0] = poww(10, tmpf.lconst[1]-1) * tmpf.lconst[1] * tmpf.errconst[1]; // determine error in A
errconst[1] = tmpf.errconst[0]; // determine error in B
```

This completes this function for this case.

Least Squares Analysis for the periodic produces a two non linear equations. Thus it can not be solved in the same manor as the power fit.

$$f_1(A, B) = \frac{\partial G}{\partial A} = -2 \sum_{k=1}^n w_k (y_k - A \cos Bx_k) \cos Bx_k = 0$$

$$f_2(A, B) = \frac{\partial G}{\partial B} = 2 \sum_{k=1}^n w_k (y_k - A \cos Bx_k) Ax_k \sin Bx_k = 0$$

There are several methods to solve such equations, and a user with the knowledge may decide to write their own code to do this. However there is a general procedure in the fitting class to solve them by a modified Newtons method. Utilizing this procedure requires providing it with two other functions. One is a function that represents the nonlinear functions to solve. For the period example the function may written as follows.

```
vec funperiodic(vec *sol, dataset *d){
vec v(2); // creates a vector called v of length 2
v.setzero(); // sets the components of v to zero
double &A = (*sol)[0]; //references the first components of the vector (*sol) as A
double &B = (*sol)[1];
try{
for (int h=0;h<d->x->no();h++) { // for loop to sum up to represent the sum part the of equations
double &x = (*d->x)[h]; // references the hth value of x in the dataset.
double &y = (*d->y)[h];
double &w = (*d->w)[h];
v[0] -= w*(y-A*cos(B*x))*cos(B*x); // represents the part inside the sum of the first components of
non linear equation.
v[1] += w*(y-A*cos(B*x))*A*x*sin(B*x); }
}catch(...) {throw(90); }
return v; }
```

The second function is required for error analysis. It represents the partial differentiation of the nonlinear equations with respect to y for one point in the dataset.

$$\frac{\partial f_1(A, B)}{\partial y} = \sum_{k=1}^n w_k \cos Bx_k$$

$$\frac{\partial f_2(A, B)}{\partial y} = \sum_{k=1}^n w_k Ax_k \sin Bx_k$$

For the periodic function it would look like

```

vec errperiodic(vec *sol,double x,double y,double w){
vec v(2);
double &A = (*sol)[0];
double &B = (*sol)[1];
try{
v[0] = w*cos(B*x);
v[1] = w*A*x*sin(B*x);
}catch(...) {throw(90); }
return v; }

```

These two functions should be written above fitl function and must take the parameters as should in the same format.

Now the fitl function can be written to employ the generic Newtons modified method function.

```

void __export customfit::fitl(bool high){
    np = 1; // number of parameters (A,B,C,D,.. etc)minus one;
    newtonsmethod(funperiodic,errperiodic);
}

```