# NUMERICAL RECIPES

## Webnote No. 19, Rev. 1

## *Code Listing fitexy*

fitexy.h

```
struct Fitexy{
```

Object for fitting a straight line $a + bx$ to a set of points $(x_i, y_i)$ with errors in both $x_i$ and $y_i$, respectively sigx and sigy. Call the constructor to calculate the fit. The answers are then available as the variables a, b, siga, sigb, chi2, and q. Output quantities a and b make $y = a + bx$ minimize $\chi^2$, whose value is returned as chi2. The $\chi^2$ probability is returned as q, a small value indicating a poor fit (sometimes indicating underestimated errors). The standard errors on a and b, siga and sigb, are not meaningful if either (i) the fit is poor, or (ii) $b$ is so large that the data are consistent with a vertical (infinite $b$) line. If siga and sigb are returned as BIG, then the data are consistent with *all* values of $b$.

```
    Doub a, b, siga, sigb, chi2, q;
```
Answers.

```
    Int ndat;
    VecDoub xx,yy,sx,sy,ww;
    Doub aa, offs;
```
Variables that communicate with Chixy.

```
    Fitexy(VecDoub_I &x, VecDoub_I &y, VecDoub_I &sigx, VecDoub_I &sigy)
    : ndat(x.size()),xx(ndat),yy(ndat),sx(ndat),sy(ndat),ww(ndat) {
```

Constructor. Call with the input data x[0..ndat-1], y[0..ndat-1], sigx[0..ndat-1], and sigy[0..ndat-1].

```
        const Doub POTN=1.571000,BIG=1.0e30,ACC=1.0e-6;
        const Doub PI=3.141592653589793238;
        Gamma gam;
        Brent brent(ACC);
        Chixy chixy(xx,yy,sx,sy,ww,aa,offs);
        Int j;
        Doub amx,amn,varx,vary,ang[7],ch[7],scale,bmn,bmx,d1,d2,r2,dum1;
        avevar(x,dum1,varx);
        avevar(y,dum1,vary);
        scale=sqrt(varx/vary);
        for (j=0;j<ndat;j++) {
            xx[j]=x[j];
            yy[j]=y[j]*scale;
            sx[j]=sigx[j];
            sy[j]=sigy[j]*scale;
            ww[j]=sqrt(SQR(sx[j])+SQR(sy[j]));
        }
        Fitab fit(xx,yy,ww);
        b = fit.b;
        offs=ang[0]=0.0;
        ang[1]=atan(b);
        ang[3]=0.0;
        ang[4]=ang[1];
        ang[5]=POTN;
        for (j=3;j<6;j++) ch[j]=chixy(ang[j]);
```

Instantiate a Chixy and bind it to our variables.

Find the $x$ and $y$ variances, and scale the data into the global variables for communication with the function chixy.

Use both $x$ and $y$ weights in first trial fit.

Trial fit for b.

Construct several angles for reference points, and make b an angle.

Bracket the $\chi^2$ minimum and then locate it with brent.

```
        brent.bracket(ang[0],ang[1],chixy);
        ang[0] = brent.ax; ang[1] = brent.bx; ang[2] = brent.cx;
        ch[0]  = brent.fa; ch[1]  = brent.fb; ch[2]  = brent.fc;
        b = brent.minimize(chixy);
```

```
        chi2=chixy(b);
        a=aa;
        q=gam.gammq(0.5*(ndat-2),chi2*0.5);
```
Compute $\chi^2$ probability.
```
        r2=0.0;
        for (j=0;j<ndat;j++) r2 += ww[j];
```
Save the inverse sum of weights at the minimum.
```
        r2=1.0/r2;
        bmx=bmn=BIG;
        offs=chi2+1.0;
```
Now, find standard errors for $b$ as points where $\Delta\chi^2 = 1$.
```
        for (j=0;j<6;j++) {
            if (ch[j] > offs) {
                d1=abs(ang[j]-b);
                while (d1 >= PI) d1 -= PI;
                d2=PI-d1;
                if (ang[j] < b) SWAP(d1,d2);
                if (d1 < bmx) bmx=d1;
                if (d2 < bmn) bmn=d2;
            }
        }
        if (bmx < BIG) {
```
Go through saved values to bracket the desired roots. Note periodicity in slope angles.
Call `zbrent` to find the roots.
```
            bmx=zbrent(chixy,b,b+bmx,ACC)-b;
            amx=aa-a;
            bmn=zbrent(chixy,b,b-bmn,ACC)-b;
            amn=aa-a;
            sigb=sqrt(0.5*(bmx*bmx+bmn*bmn))/(scale*SQR(cos(b)));
            siga=sqrt(0.5*(amx*amx+amn*amn)+r2)/scale;
```
Error in a has additional piece `r2`.
```
        } else sigb=siga=BIG;
        a /= scale;
```
Unscale the answers.
```
        b=tan(b)/scale;
    }

    struct Chixy {
```
Captive functor of `Fitexy`, returns the value of $(\chi^2 - \texttt{offs})$ for the slope `b=tan(bang)`. Scaled data and `offs` are communicated via bound references.
```
        VecDoub &xx,&yy,&sx,&sy,&ww;
        Doub &aa,&offs;

        Chixy(VecDoub &xxx, VecDoub &yyy, VecDoub &ssx, VecDoub &ssy,
        VecDoub &www, Doub &aaa, Doub &ooffs) : xx(xxx),yy(yyy),sx(ssx),
        sy(ssy),ww(www),aa(aaa),offs(ooffs) {}
```
Constructor. Bind references back to `Fitexy`.
```
        Doub operator()(const Doub bang) {
```
The function as seen by `Brent` and `zbrent`.
```
            const Doub BIG=1.0e30;
            Int j,nn=xx.size();
            Doub ans,avex=0.0,avey=0.0,sumw=0.0,b;
            b=tan(bang);
            for (j=0;j<nn;j++) {
                ww[j] = SQR(b*sx[j])+SQR(sy[j]);
                sumw += (ww[j]=(ww[j] < 1.0/BIG ? BIG : 1.0/ww[j]));
                avex += ww[j]*xx[j];
                avey += ww[j]*yy[j];
            }
            avex /= sumw;
            avey /= sumw;
            aa=avey-b*avex;
            for (ans = -offs,j=0;j<nn;j++)
                ans += ww[j]*SQR(yy[j]-aa-b*xx[j]);
            return ans;
        }
    };

};
```