FIT TO INITIALLY LINEAR SECTION WITH SATURATION

RICHARD J. MATHAR

Abstract. The solution of the quadratic fit to a set of data points \((x_i, y_i)\)
with the standard least-squares minimization of the errors in the \(y\)-values is
derived, then the solution of a fit with a rational polynomial with numerator
and denominator having both degree one.

A C++ program which implements both for data pairs read from a file is
provided.

1. Aim

A set of \(N\) data pairs \((x_i, y_i)\) in the index range \(0 \leq i < N\) is supposed to be
given, assuming that these are samples of a curve which is almost linear but contains
some tendency to ‘saturation.’ We derive the three parameters which represent a
least-squares fit by a quadratic polynomial

\[
y = \alpha_0 + \alpha_1 x + \alpha_2 x^2,
\]

or alternatively by a rational polynomial

\[
y = \frac{a_0 + a_1 x}{1 + b_1 x}.
\]

2. Polynomial Fits

For a quadratic fit, we minimize the sum over all squared vertical deviations of
the data points \((x_i, y_i)\) to the curve,

\[
\sum_i |\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i|^2 \rightarrow \min.
\]

The partial derivatives with respect to \(\alpha_0\), \(\alpha_1\) and \(\alpha_2\) are set to zero to locate the
minimum in the parameter range:

\[
\frac{\partial}{\partial \alpha_0} \sum_i |\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i|^2 = 0;
\]

\[
\frac{\partial}{\partial \alpha_1} \sum_i |\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i|^2 = 0;
\]

\[
\frac{\partial}{\partial \alpha_2} \sum_i |\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i|^2 = 0.
\]

Date: December 18, 2012.

Key words and phrases. Polynomial Fit.
Building exterior and interior derivatives, the division of each equation through 2 gives

\[ \sum_i (\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i) = 0; \quad (7) \]
\[ \sum_i x_i (\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i) = 0; \quad (8) \]
\[ \sum_i x_i^2 (\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i) = 0. \quad (9) \]

This is a 3\times3 inhomogeneous linear system of equations with a symmetric coefficient matrix for the three unknowns \(\alpha\):

\[ \begin{pmatrix} N \sum_i x_i & \sum_i x_i^2 & \sum_i x_i^3 \\ \sum_i x_i^2 & \sum_i x_i^3 & \sum_i x_i^4 \\ \sum_i x_i^3 & \sum_i x_i^4 & \sum_i x_i^5 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} \sum_i y_i \\ \sum_i x_i y_i \\ \sum_i x_i^2 y_i \end{pmatrix}. \quad (10) \]

Note that the well-known linear fit is obtained if the components associated with \(\alpha_2\), which is the bottom row and right column of the matrix and the bottom component of the right hand side, are deleted.

In the numerical implementation, the 4 sums \(\sum_i x_i^k\) for \(k = 1, \ldots, 4\) and the 3 mixed sums \(\sum_i x_i^k y_i\) for \(k = 0 \ldots 2\) are gathered. The system of equations is then solved either by Gaussian elimination, by Cramer’s rule or even by inversion of the matrix. The function \texttt{fitsqr} in Appendix A is an implementation of this methodology in C++.

The fit to polynomials of higher degree follows exactly the same path. The symmetric matrix of the system of linear equations increases one by one, and the powers of \(x_i\) in the matrix and in the vector of the right hand side also rise regularly [1, 3]. The NICMOS data handbook uses cubic polynomials, for example. Equivalent implementations are in the Interpolation library of the GNU Scientific Library.

The equation (10) says that the parameters \(\alpha\) are the inverse \(X^{-1}\) of a matrix that depends on the abscissa values \(x\) (which are supposed to be error-free) multiplied by a vector of dot products of the \(x\) and the \(y\). The covariance matrix of the estimates \(\alpha\) (and basically the estimated errors) is therefore the covariance of the vector of the right hand side with the elements of \(X^{-1}\) introduced as weights for the powers of \(x_i\).

3. Fit With Saturation

3.1. Padé Form. A curve with three parameters \(a_0, a_1\) and \(b_1\) of the form

\[ y = \frac{a_0 + a_1 x}{1 + b_1 x} \quad (11) \]

suits as a model of an S-shaped curve with intersection \(a_0\) with the vertical axis, initially linear behaviour

\[ y \approx a_0 + (a_1 - a_0 b_1)x, \quad x \to 0 \quad (12) \]

and a saturation value of

\[ y \to a_1/b_1, \quad x \to \infty. \quad (13) \]

It is a Hill function with exponent 1.
A conceptually simple way of approximating this rational function \( y(x) \) is to look at it as a low-order Padé approximation of the associated quadratic fit, and to calculate first the quadratic fit to the data points and refitting the result \([2]\),

\[
y = a_0 + (a_1 - a_0 b_1) x + (-a_1 + a_0 b_1) b_1 x^2 + O(x^3). \tag{14}
\]

The disadvantage of this approach is that the least-squares minimization property is lost. The advantage is that it can easily be extended to higher order polynomials (of the same degree, i.e., on the diagonal of the Padé table) in numerator and denominator.

### 3.2. Nonlinear Least Squares.

#### 3.2.1. Nonlinear System of Equations.

The parameters that fit a set of data pairs \((x_i, y_i)\) by \((11)\) in the least squares sense are given by setting the 3 partial derivatives \(\partial/\partial a_0, \partial/\partial a_1, \partial/\partial b_1\), of

\[
\sum \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right)^2 \rightarrow \text{min}, \tag{15}
\]

to zero, generating the nonlinear system of equations

\[
\begin{align*}
\sum \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{1}{1 + b_1 x_i} &= 0; \tag{16} \\
\sum \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{x_i}{1 + b_1 x_i} &= 0; \tag{17} \\
\sum \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \left( \frac{a_0 + a_1 x_i}{(1 + b_1 x_i)^2} \right) &= 0. \tag{18}
\end{align*}
\]

#### 3.2.2. Numerical Solutions.

**Remark 1.** A strategy to solve this system is to assume \(b_1\) is known from an initial estimate (say, \(b_1 = 0\)), solving the first two equations as a symmetric linear system of two unknowns,

\[
\begin{pmatrix}
\sum \frac{1}{(1 + b_1 x_i)^2} & \sum \frac{x_i}{(1 + b_1 x_i)^2} \\
\sum \frac{x_i}{(1 + b_1 x_i)^2} & \sum \frac{x_i^2}{(1 + b_1 x_i)^2}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1
\end{pmatrix}
= \begin{pmatrix}
\sum \frac{y_i}{1 + b_1 x_i} \\
\sum \frac{y_i x_i}{1 + b_1 x_i}
\end{pmatrix}.
\]

solving the third equation given these values of \(a_0, a_1\) for \(b_1\), and iterating this process to convergence.

The nonlinear equation for \(b_1\) may be addressed with the Newton iteration method, calculating updates \(b_1 \rightarrow b_1 + \Delta b_1\) with

\[
\sum \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \left( \frac{a_0 + a_1 x_i}{(1 + b_1 x_i)^2} \right) + A_{22} \Delta b_1 = 0. \tag{20}
\]

where

\[
A_{22} \equiv 3 \sum \frac{x_i^2 (a_0 + a_1 x_i)^2}{(1 + b_1 x_i)^4} - 2 \sum \frac{y_i x_i^2 (a_0 + a_1 x_i)}{(1 + b_1 x_i)^3} = \sum \frac{x_i^2 (a_0 + a_1 x_i)^2}{(1 + b_1 x_i)^3} \left[ 3 \frac{(a_0 + a_1 x_i)}{1 + b_1 x_i} - 2 y_i \right]. \tag{21}
\]

It turns out that the convergence of this method of alternating solutions for \((a_0, a_1)\) and \(b_1\) is slow.
The Newton-Raphson implementation looks as follows: From an initial estimate, for example the one from the previous remark, update \(a_0 \rightarrow a_0 + \Delta a_0\), \(a_1 \rightarrow a_1 + \Delta a_1\), \(b_1 \rightarrow b_1 + \Delta b_1\), where

\[
\begin{align*}
\sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{1}{1 + b_1 x_i} + A_{00} \Delta a_0 + A_{01} \Delta a_1 + A_{02} \Delta b_1 &= 0; \\
\sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{x_i}{1 + b_1 x_i} + A_{10} \Delta a_0 + A_{11} \Delta a_1 + A_{12} \Delta b_1 &= 0; \\
\sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{a_0 + a_1 x_i}{(1 + b_1 x_i)^2} + A_{20} \Delta a_0 + A_{21} \Delta a_1 + A_{22} \Delta b_1 &= 0,
\end{align*}
\]

is an inhomogeneous linear system for the vector of the three unknown corrections \(\Delta a_0\), \(\Delta a_1\) and \(\Delta b_1\), and where

\[
\begin{align*}
A_{00} &\equiv \frac{\partial}{\partial a_0} \sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{1}{1 + b_1 x_i} = \sum_i \frac{1}{(1 + b_1 x_i)^2}; \\
A_{11} &\equiv \frac{\partial}{\partial a_1} \sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{x_i}{1 + b_1 x_i} = \sum_i \frac{x_i}{(1 + b_1 x_i)^2}; \\
A_{01} &= A_{10} \equiv \frac{\partial}{\partial a_1} \sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{1}{1 + b_1 x_i} = \sum_i \frac{x_i}{(1 + b_1 x_i)^2}; \\
A_{02} &= A_{20} \equiv \frac{\partial}{\partial b_1} \sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{1}{1 + b_1 x_i} = \sum_i \frac{x_i}{(1 + b_1 x_i)^2} \left[ -2 \frac{a_0 + a_1 x_i}{1 + b_1 x_i} + y_i \right]; \\
A_{12} &= A_{21} \equiv \frac{\partial}{\partial b_1} \sum_i \left( \frac{a_0 + a_1 x_i}{1 + b_1 x_i} - y_i \right) \frac{x_i}{1 + b_1 x_i} = \sum_i \frac{x_i^2}{(1 + b_1 x_i)^2} \left[ -2 \frac{a_0 + a_1 x_i}{1 + b_1 x_i} + y_i \right],
\end{align*}
\]

build the \(3 \times 3\) coefficient matrix of the second derivatives of the function to be minimized. \(A_{22}\) is already defined above. The function \texttt{fitsat}\ in Appendix A is an implementation of this methodology in C++.

APPENDIX A. IMPLEMENTATION OF THE FITS

A.1. File \texttt{fitSqrMain.hxx}.

```cpp
#include <cstring>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <cmath>
#include "fitSqr.h"

using namespace std;

/** Main demonstrator function. 
* The program is called with one argument, which is an existing 
* ASCII file. This file contains either lines starting with #, and these 
* lines are ignored. Other lines contain each two floating point numbers, 
* separated by white space, which define one pair of x and y. 
* 
* The function prints the 3 coefficients of the quadratic fit, the mean
```
If the scattered data points (red) to be fitted reach into the saturation regime, the quadratic fit (green) faces the risk of relocating its local maximum into the interval of the fit.

If the data cover more of the linear range and stay away from the saturation regime, the rational (blue) and quadratic (green) fit become more and more the same.
int main(int argc, char *argv[])
{
    /* the current line of the input file */
    char inl[256] ;
    /* number of input pairs */
    int N=0 ;
    /* the three coefficients of the fit */
    double lincoef[3] ;
    /* variances of the three coefficients of the fit */
    double lincoefVar[3] ;
    /* sum of squared deviations */
    double rms ;

double *x = (double *)malloc(0) ;
double *y = (double *)malloc(0) ;
if ( argc != 2)
{
    cerr << "Usage: " << argv[0] << " inputfile.asc" << endl ;
    return 1 ;
}

FILE * f = fopen(argv[1],"r") ;
while( fgets(inl,256,f) )
{
if( inl[0] != '#')
{
    double xy[2] ;
    if ( sscanf(inl,"%lf %lf",xy[0],&xy[1]) == 2)
    {
        N++ ;
        x= (double*)realloc(x,N*sizeof(double)) ;
        y= (double*)realloc(y,N*sizeof(double)) ;
        x[N-1] = xy[0] ;
    }
}
fclose(f) ;
}
/* do the quadratic fit */
rms = fitSqr::fitsqr(x,y,N,lincoef, lincoefVar) ;
cout << "# poly mean devs " << sqrt(lincoefVar[0]/(N-2)) << " " << sqrt(lincoefVar[1]/(N-2))
     << " " << sqrt(lincoefVar[2]/(N-2)) << endl;
cout << "# mean error " << sqrt(rms/(N-2)) << endl ;

/* do the s-shaped fit with saturation. Keep lincoef as a first estimate */
rms = fitSqr::fitsat(x,y,N,lincoef) ;
cout << "# mean error " << sqrt(rms/(N-2)) << endl ;

/* free resources bundled above */
free(x) ;
free(y) ;
return 0 ;
} /* main */

A.2. File fitSqr.h.
 ifndef FITSQR_H
#define FITSQR_H

class fitSqr {
  public:
    fitSqr() ;

    static double det33(const double matr[3][3]) ;
    static void solve33(const double A[3][3], const double rhs[3], double res[3]) ;
    static void invert33(const double A[3][3], double inv[3][3]) ;
    static double sqrVar(const double x[], const double y[], int N, double alpha[3], int p) ;
    static double fitsqr(const double x[], const double y[], int N, double alpha[3], double alphaVar[]) ;
    static double fitsat(const double x[], const double y[], int N, double alpha[3]) ;

  protected:
    private:
  } ;

#endif /* FITSQR_H */

A.3. File fitSqr.cxx.
 #include <string.h>
 #include <stdio.h>
 #include <stdlib.h>
 #include <math.h>
 #include <math.h>
#include "fitSqr.h"

/** Determinant of the 2 by 2 matrix */
#define DET22(a00,a01,a10,a11) ((a00)*(a11)-(a01)*(a10))

/** Determinant of a 3 by 3 matrix */
#define DET33(matr) (((matr[0][0]*(matr[1][1]*matr[2][2]-matr[1][2]*matr[2][1]))-
(matr[0][1]*(matr[1][0]*matr[2][2]-matr[1][2]*matr[2][0]))+
(matr[0][2]*(matr[1][0]*matr[2][1]-matr[1][1]*matr[2][0])))

/** Solve a 3 by 3 system of linear equations. */
#define SOLVE33(A,rhs,res) {
  double cramdenom = det33(A) ;
  double cram[3][3] ;
  for(int c=0 ; c < 3 ; c++) {
    memcpy(cram,A,9*sizeof(double)) ;
    for(int r=0 ; r < 3 ; r++)
      cram[r][c] = rhs[r] ;
    res[c] = det33(cram)/ cramdenom ;
  }
}

/** Invert a 3 by 3 matrix. */
#define INVERT33(A) {
  SOLVE33(A,A,A) ;
}

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void) {
  double A[3][3] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 };
  double rhs[3] = { 10.0, 11.0, 12.0 };
  double res[3];
  INVERT33(A);
  for(int i=0; i < 3; i++)
    printf("%f\n", res[i]);
  return 0;
}
void fitSqr::invert33(const double A[3][3], double inv[3][3])
{
    /* denominator in Cramer’s rule */
    double cramdenom = det33(A) ;

    /* Matrix with the substituted columns for Cramer’s rule */
    double cram[3][3] ;
    for(int p=0 ; p < 3 ; p++)
    {
        /* determine the three solutions alpha by Cramer’s rule */
        for(int c=0 ; c < 3 ; c++)
        {
            memcpy(cram,A,9*sizeof(double)) ;
            /* push right hand side into column c of the matrix */
            for(int r=0 ; r< 3; r++)
            {
                cram[r][c] = (r==p) ? 1.0 : 0.0 ;
            }
            inv[c][p] = det33(cram)/ cramdenom ;
        }
    }
}

double fitSqr::sqrVar(const double x[], const double y[], int N, double alpha[3], int p)
{
    double v=0. ;
    switch(p)
    {
    case 0 :
        for (int i=0 ; i < N ; i++)
            v += pow( alpha[0]+x[i]*(alpha[1]+alpha[2]*x[i])-y[i],2.) ;
        break;
    default :
        for (int i=0 ; i < N ; i++)
            v += pow(x[i],(double)p)*pow(alpha[0]+x[i]*(alpha[1]+alpha[2]*x[i])-y[i],2.) ;
        break;
    }
    return v;
}

*/
```cpp
/** Fit a set of scattered data points (x,y) with a quadratic polynomial.
 * @param[in] x The vector of all x-values. has N components.
 * @param[in] y The vector of all y-values. has N components.
 * @param[in] N The number of components individually in x and y.
 * @param[out] alpha The three coefficients of the quadratic fit.
 * @param[out] alphaVar The main diagonal of the variance matrix of the alpha.
 * @return Sum of the squares of the deviations from the fit.
 */

double fitSqr::fitsqr(const double x[], const double y[], int N, double alpha[3], double alphaVar[3])
{
    /* 3 by 3 matrix of the inhomogeneous linear system of eqs */
    double A[3][3] ;

    /* Inverse of the 3 by 3 matrix of the inhomogeneous linear system of eqs */
    double Ainv[3][3] ;

    /* Matrix with the substituted columns for Cramer's rule */
    double cram[3][3] ;

    /* right hand side of the system of equations */
    double rhs[3] ;

    /* variances of the right hand side of the system of equations */
    double rhsVar[3][3] ;

    /* denominator in Cramer's rule */
    double cramdenom ;

    A[0][0] = N ;
    /* sum over all x_i and y_i */
    for (int i=0 ; i < N ; i++)
    {
        const double xsq = x[i]*x[i] ;
        A[0][1] += x[i] ;
        A[0][2] += xsq ;
        A[1][2] += x[i]*xsq ;
        A[2][2] += xsq*xsq ;
        rhs[0] += y[i] ;
        rhs[1] += y[i]*x[i] ;
        rhs[2] += y[i]*xsq ;
    }
    A[1][0] = A[0][1] ;
```
/* solve the inhomogeneous linear system */
solve33(A,rhs,alpha) ;

/* Calculate the variances by multiplication of the variances of the rhs by the inverse matrix. Uses the symmetry in the covariance matrix. */
invert33(A,Ainv) ;

for(int r=0 ; r < 3; r++)
for(int c=r ; c < 3; c++)
    rhsVar[c][r] = rhsVar[r][c] = sqrVar(x, y, N, alpha, r+c) ;

#if 0
/* print the covariance matrix. Not useful in general */
printf("rhsVar %e %e %e
",rhsVar[0][0],rhsVar[0][1],rhsVar[0][2]) ;
printf("rhsVar %e %e %e
",rhsVar[1][0],rhsVar[1][1],rhsVar[1][2]) ;
printf("rhsVar %e %e %e
",rhsVar[2][0],rhsVar[2][1],rhsVar[2][2]) ;
/* print the inverse matrix. Not useful in general */
printf("inv %e %e %e
",Ainv[0][0],Ainv[0][1],Ainv[0][2]) ;
printf("inv %e %e %e
",Ainv[1][0],Ainv[1][1],Ainv[1][2]) ;
printf("inv %e %e %e
",Ainv[2][0],Ainv[2][1],Ainv[2][2]) ;
#endif

/* take only care of the main diagonal of the covariance. No PVD yet. */
for(int r=0 ; r < 3; r++)
{
    alphaVar[r] = 0. ;
    for(int i=0 ; i < 3; i++)
    for(int j=0 ; j < 3; j++)
        alphaVar[r] += Ainv[r][i]*Ainv[r][j]*rhsVar[i][j] ;
}

/* return the sum of all squared deviations */
return rhsVar[0][0] ;

/** Fit a set of scattered data points (x,y) to a first-order rational polynomial. */
** @param[in] x The vector of all x-values. has N components.
** @param[in] y The vector of all y-values. has N components.
** @param[in] N The number of components individually in x and y.
** @param[out] alpha The three coefficients of the fit.
** @return Sum of the squares of the deviations from the fit. */
double fitSqr::fitsat(const double x[], const double y[], int N, double alpha[3])
{
    /* 3 by 3 matrix of the inhomogeneous linear syste of eqs */
    double A[3][3] ;
/* right hand side of the system of equations */ double rhs[3] ; /* denominator in Cramer's rule */ double cramdenom ; /* Matrix with the substituted columns for Cramer's rule */ double cram[3][3] ; /* relaxation factor in the newton-ralphson iterations */ double rlxf = 0.30 ; /* re-interpret the coefficients of the quadratic fit * in terms of the rational function fit */ alpha[2]= -alpha[2]/alpha[1] ; alpha[1] += alpha[0]*alpha[2] ; #ifdef DEBUG printf("# start %e %e %e\n",alpha[0],alpha[1],alpha[2]) ; #endif for(int itr =0 ; itr<30; itr++) { for(int r=0 ; r < 3 ; r++) A[r][0] = A[r][1] = A[r][2] = rhs[r] = 0. ; /* to stabilize the method, use only half, then 3/4 and eventually * all of the data values to fit the three parameters. */ int Neff = ( itr < 4 ) ? N/2 : ( itr< 8 ? 3*N/4 : N) ; /* build matrix and right hand side by summation over all noisy values */ for (int i=0 ; i < Neff ; i++) { const double xinv = 1/(1+alpha[2]*x[i]) ; const double xinv2 = xinv*xinv ; const double xlin = alpha[0]+alpha[1]*x[i] ; const double xlinr = xlin*xinv-y[i] ; rhs[0] -= xlinr*xinv ; rhs[1] -= xlinr*xinv*x[i] ; rhs[2] += xlinr*xlin*x[i]*xinv2 ; A[0][0] += xinv2 ; A[0][1] += x[i]*xinv2 ; A[0][2] += x[i]*xinv2*(-2.*xlin*xinv+y[i]) ; A[1][1] += x[i]*x[i]*xinv2 ; A[1][2] += x[i]*x[i]*xinv2*(-2.*xlin*xinv+y[i]) ; A[2][2] += x[i]*x[i]*x[i]*xinv2 ; } }
A[2][2] += x[i]*x[i]*xlin*xinv2*(3.*xlin*xinv2-2*y[i]*xinv) ;
}
A[1][0] = A[0][1] ;
#endif DEBUG

printf("# zero ? %e %e %e\n",rhs[0],rhs[1],rhs[2]) ;
#endif

/* denominator in Cramer's rule */
cramdenom = det33(A) ;

/* determine the three solutions alpha by Cramer’s rule */
for(int c=0 ; c < 3 ; c++)
{
    memcpy(cram,A,9*sizeof(double)) ;
    /* push right hand side into column c of the matrix */
    for(int r=0 ; r< 3; r++)
        cram[r][c] = rhs[r] ;
    alpha[c] += rlxf *det33(cram)/ cramdenom ;
}
if ( itr > 15)
    rlxf = 0.5 ;
#endif DEBUG

/* if in debugging mode, print the some progress */
printf("# alph = %e %e %e rl= %f\n",alpha[0],alpha[1], alpha[2],rlxf) ;
#endif

/* calculate the sum of all squared deviations */
cramdenom=0. ;
for(int i=0 ; i < N ; i++)
{
    cramenom += pow( (alpha[0]+x[i]*alpha[1])/(1.+alpha[2]*x[i])-y[i],2.) ;
#endif DEBUG

    /* if in debugging mode, print the original values and their */
    /* associated values on the fitting curve, suitable for gnuplot. */
    printf("%e %e %e\n",x[i],y[i], (alpha[0]+x[i]*alpha[1])/(1.+alpha[2]*x[i]) ) ;
#endif
}

return cramenom ;
} /* fitsat */
References


E-mail address: mathar@mpia.de
URL: http://www.mpia.de/~mathar

Max-Planck Institute of Astronomy, Königstuhl 17, 69117 Heidelberg, Germany