PowFit2D

Richard J. Mathar

Generated by Doxygen 1.8.2

Mon Oct 15 2012 12:17:39

Contents

1 Main Page

1.1 Usage

```
PowFit2D [-v] [-p maxdegree] inputfilename > outputfile
```

The `inputfilename` is a list of function values on a finite discrete set of 2D Cartesian coordinates as described in PowFit2D::readf(). The standard output contains the fitting coefficients as described in PowFit2D::display(). If one wants to delete the comment lines that are dispersed in the standard output, one can use a pipe like

```
PowFit2D [-v] [-p maxdegree] inputfilename | fgrep -v '#' | awk '{print
      $1 $2 $3}' > outputfile
```

One can extract the original input and the fit by using the -v option and looking at the lines between FITTED START and FITTED END:

```
PowFit2D -v inputfilename | sed -n '/FITTED START/,/FITTED END/p' | sed
      '/FITTED/d' | tr "#" " " > outputfile
```

and this output is immediately useful for gnuplot(1)'s splot command.

```
gnuplot
gnuplot> splot "outputfile" title "original", "outputfile" using 1:2:4 title "
      fitted"
gnuplot> quit
```

1.1.1 line options

See PowFit2D::main() .

1.2 Description

See PowFit2D::fit() for the fitting formul.

1.3 Compilation

A prerequisite is that the LAPACK library of netlib.org has been installed.

```
g++ -o PowFit2D -I<dir_of_clapack.h> PowFit2D.cpp -
      L<dir_of_liblapack> -llapack
```

The makefile entry is

```
LIBFLAGS=-L<dir_of_liblapack>
INCFLAGS=-I<dir_of_clapack.h>
PowFit2D: PowFit2D.cpp
     $(CXX) -o $@ $(INCFLAGS) $< $(LIBFLAGS) -llapack
```

IF the GSL library is available, this ought be indicated by setting preprocessor variable HAVE_GSL, then the LAPACK library is not needed

```
g++ -o PowFit2D -I<dir_of_gsl> -DHAVE_GSL PowFit2D.cpp -L... -
      lgslcblas -lgsl
```

Author

Richard J. Mathar

Todo compute error bars on the fitting coefficients

Since

2007-07-12 Richard J. Mathar homepage.

See Also

[GuoOptik117]}

2 Todo List

page Main Page

compute error bars on the fitting coefficients

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Data Structure Documentation

5.1 PowFit2D Class Reference

Public Member Functions

- PowFit2D (char ∗fname, const bool verbos=false)
- void fit (int maxP=-1)
- void display (const bool verb=false)

Private Member Functions

- double fitted (const double x, const double y)
- void readf (char ∗fname, bool verb=false)
- int triang (const int i)
- int strIndx (const int i, const int j)
- void strIndxInv (const int n, int &i, int &j)

Static Private Member Functions

- static bool isComment (const string inpli, const regex_t &preg)
- static double binomial (int a, int b)
- static double cosFlatn (const int l, const int k, int m)

Private Attributes

- vector< double > x
- vector< double > y
- vector< double > f
- bool verb
- vector< double > alpha
- int maxHybrP

5.1.1 Detailed Description

The class contains a list of 2D points (x_k, y_k), $k = 1, .., N$, and scalar functions values f_k of these, and performs a least squares fit on a polynomial basis in the sense of $f_k \approx \sum_{i,j} \alpha_{i,j} x_k^i y_k^j$.

Since

2007-07-12

5.1.2 Constructor & Destructor Documentation

5.1.2.1 PowFit2D::PowFit2D (char * *fname*, const bool *verbos* = `false`) `[inline]`

Ctor. This initializes the PowFit2D::x, PowFit2D::y and PowFit2D::f vectors with the triples of floating point numbers that are to be fitted.

Parameters

in	*fname*	the file name of the input file with the tabulated (x,y,f) triples.

Since

2007-07-12

```
                                                : verb(verbos)
{
    readf(fname,verbos) ;
}
```

5.1.3 Member Function Documentation

5.1.3.1 static double PowFit2D::binomial (int *a*, int *b*) `[inline]`, `[static]`, `[private]`

The binomial C(a,b). [**?** , 3.1.2]

Parameters

in	*a*	the non-negative numerator of the binomial
in	*b*	the non-negative denomiator, less or equal to a

Returns

the binomial coefficient $C(a,b) = a!/b!/(a-b)!$

Since

2007-07-12

See Also

OEIS A007318

```
    {
#ifdef HAVE_GSL
        return gsl_sf_choose(unsigned(a),unsigned(b)) ;
#else

        if ( a<0 || b<0 || b>a)
            return 0. ;
        if (a-b < b)
            b = a-b ;
```

```
        if ( b == 0 )
            return 1. ;
        double resul = a ;
        while ( b-- > 1)
            resul *= double(a-b+1)/double(b) ;
        return a ;
#endif /* HAVE_GSL */
    }
```

5.1.3.2 static double PowFit2D::cosFlatn (const int *l*, const int *k*, int *m*) [inline], [static], [private]

The contribution of $\cos^l(\varphi)\sin^k(\varphi)$ to $\cos(m\varphi)$ or $\sin(m\varphi)$. With the Euler formula [**?** , 4.3] and the binomial expansion [**?** , 3.1.3] we have $\cos^l\varphi\sin^k\varphi = (e^{i\varphi}+e^{-i\varphi})^l/(2^l)(e^{i\varphi}-e^{-i\varphi})^k/(2i)^k \sim \frac{1}{2^{k+l}}(-)^{\langle k/2\rangle}\sum_{s=0}^{l}\binom{l}{s}\sum_{t=0}^{k}\binom{k}{t}(-)^t e^{i(2s-l+k-2t)\varphi}$ to be divided by i if k is odd. The algorithm is simply to match the integer in the exponential with m.

Parameters

in	*l*	the power of the cosine
in	*k*	the power of the sine
in	*m*	the cycle number of the cosine (m>=0) or the sine (m<0) for which the coefficient is created.

Returns

the parameter beta in the expansion cos^l phi sin^k phi = ... beta∗cos(m∗phi) or = .. beta∗sin(|m|∗phi)

Since

2007-07-12

```
{
    double resul=0. ;
    /* for nonzero result, an even k must meet a cosine expansion
   coeffieicnt
    * and an odd k must meet a sine expansion coefficient.
    */
    if ( m >=0  && k % 2 == 0  || m < 0 && k % 2)
    {
        m = abs(m) ;

        /* k/2 if k is even , floor(k/2) if k is odd
        */
        const int khalf = k /2 ;
        if ( (m+l+k) % 2 ==0)
        {
            for(int s=0 ; s <= l ; s++)
            {
                const int t= s-(m+l-k)/2 ;
                if ( t>=0 && t <= k)
                    if ( t % 2 )
                        resul -= binomial(l,s)*binomial(k,t
  ) ;
                    else
                        resul += binomial(l,s)*binomial(k,t
  ) ;
            }
            if ( khalf % 2)
                resul *= -1. ;
            resul /= pow(2.,l+k) ;
        }
    }
    return resul ;
}
```

5.1.3.3 void PowFit2D::display (const bool *verb* = false) [inline]

Show the fitted function on stdout. The output format is:

- lines that start with hashes (#) are comment lines. Their number depends on whether the -v command line option was used or not.

  - If the -v option was used, they contain a list of the original (x,y,f) triples, each time followed by the value of the fitted bivariate and the deviation between the fit and the original f. This optional section is started with a line POWFIT2D FITTED START and continues up to a line that contains POWFIT2D FITTED END.
  - The comment lines contain a section started with POWFIT2D SPHERICAL START and concluded with POWFIT2D SPHERICAL END which show the fitting function transformed to spherical coordinates, $x = r\cos(\varphi)$, $y = r\sin(\varphi)$. Each of these lines is a term in the expansion $f = \sum \beta_k r^p (\sin,\cos)(m\varphi)$ encoded as follows: the first number is the integer p. The second number is the value of m. If this value is positive or zero, the term is $\propto \cos(m\varphi)$, if this value is negative, the term is $\propto \sin(|m|\varphi)$. The third number is the value of β for this term, which may be zero. This functionality is equivalent to a lookup via the table in `Zernike.txt`.

- For lines that are not comment lines, the first non-negative integer is the power i in the x coordinate, the next non-negative integer is the power j in the y coordinate, and the following number is the expansion coefficient $\alpha_{i,j}$ in $f \approx \sum_{i,j} \alpha_{i,j} x^i y^j$. The rest of the line is an expression summarizing this in more human-readable fashion.

Warning

this only makes sense if PowFit2D.fit() had been called before. [in] verb verbosity level

Since

2007-07-12

```
{
    /* Display the main result, which is the expansion coefficients
    * and their associated powers in the two cartesian coordinates.
    */
    for(int i=0; i < alpha.size() ; i++)
    {
        int powx, powy ;
        strIndxInv(i,powx,powy) ;
        cout << powx << " " << powy << " " << alpha[i] << " "
            << alpha[i] << "*x^" << powx << "*y^" << powy << endl ;
    }

    /* If verbose output is demanded, also show the fit of the quality
    * by providing the original triples together with their fitted
    * approximation by the bi-variate polynomial and the error between
    * the fit and the input value.
    */
    if ( verb)
    {
        cout << "# POWFIT2D FITTED START\n" ;
        for(int i=0; i < x.size() ; i++)
        {
            const double fi = fitted(x[i],y[i]) ;
            cout << "# " << x[i] << " " << y[i] << " " << f[i] << " " <<
  fi << " " << fi-f[i] << endl ;
        }
        cout << "# POWFIT2D FITTED END\n" ;
    }

    /* Finally convert the expansion coefficients in the Cartesian
    * coordinates to expansion coefficients in the spherical system.
    */
    cout << "# POWFIT2D SPHERICAL START x=r*cos phi, y=r*sin phi\n" ;

    /* loop over the powers r^p
    */
    for(int p=0; p <= maxHybrP ; p++)
    {
        /* loop over the sin(m*phi) with m<0 and the cos(m*phi) with m>=0.
        */
        for(int m= p; m >= -p ; m--)
        {
            double co = 0. ;
            /* gather the contribution of all mixed powers individually.
            * The powx+powy=p that contribute with x^powx*y^powy are all in
  a consecutiave
            * range of straightened indices.
            */
            for(int s= triang(p) ; s < triang(p+1) ; s++)
            {
                int powx, powy ;
                strIndxInv(s,powx,powy) ;
```

```
                const int degfact = ( m==0) ? 1 : 2 ;
                co += degfact*alpha[s]*cosFlatn(powx,powy,m) ;
            }
            cout << "# " << p << " " << m << " " << co << " " << co << "*r^
    " << p ;
            if ( m > 0)
                cout << "*cos(" << m << "*phi)" << endl ;
            else if ( m < 0 )
                cout << "*sin(" << -m << "*phi)" << endl ;
            else
                cout << endl ;
        }
    }
    cout << "# POWFIT2D SPHERICAL END\n" ;
}
```

5.1.3.4 void PowFit2D::fit (int *maxP* = -1) [inline]

Do the fitting. Minimize $\sum_k [f(x_k,y_k) - f_k]^2 \rightarrow \min.$ with the ansatz $f(x,y) = \sum_{p=0}^{n} \sum_{i=0}^{p} \alpha_{i,p-i} x^i y^{p-i}$. which leads to the linear system of equations for the unknown $\alpha_{i,p-i}$ $\sum_{p,i} \alpha_{i,p-i} \sum_k x_k^i y_k^{p-i} x_k^l y_k^m = \sum_k f_k x_k^l y_k^m$.

Parameters

in	*maxP*	the maximum power in the fitting procedure. The default of -1 means that it is chosen automatically to be the maximum allowed by the number of 2D points that have been read in.

Since

2007-07-12

```
{
    /* the degrees of freedom as determined by the number of points.
    */
    const int deg = x.size() ;

    if( deg <=0)
    {
        cerr << "Number of points " << deg << " insufficient\n";
        exit(EXIT_FAILURE) ;
    }

    /* determine maxP automatically. deg=0 does not allow fitting.
    * deg=1,2 means maxP=0, deg=3..5 means maxP=1 etc.
    */
    if ( maxP < 0 )
    {
        maxP = 0 ;
        while ( triang(maxP+2) <= deg)
            maxP++ ;
    }
    else
    {
        while ( triang(maxP+1) > deg)
            maxP-- ;
    }

    maxHybrP = maxP ;

    /* the number of fitting degrees of freedom
    * We keep this an overdetermined system, n<=deg. The case of n=deg
    * means the fit is actually an interpolation (ie, exact).
    */
    int n=triang(maxP+1) ;

    if (verb)
    {
        cout << "# maximum total power " << maxP << " for " << deg << "
   input points; "
             << n << " fitting degrees\n" ;
    }

#ifdef HAVE_GSL
    /* the n by n matrix of the linear system of equations */
    gsl_matrix *A=gsl_matrix_alloc(n,n) ;
    gsl_matrix_set_all(A,0.) ;

    /* the RHS of the linear system of equations */
```

```
        gsl_vector *rhs=gsl_vector_alloc(n) ;
        gsl_vector_set_all(rhs,0.) ;

        /* fill the matrix and the right hand side
        */
        for(int row=0; row<n ;row++)
        {
            int rowi, rowj ;
            strIndxInv(row,rowi,rowj) ;
            for(int col=0; col< n ;col++)
            {
                int coli, colj ;
                strIndxInv(col,coli,colj) ;
                /* debugging
                * cout << "# mapped " << row << " " << rowi << " " << rowj <<
     endl ;
                */
                for(int k=0 ; k < deg ;k++)
                    * gsl_matrix_ptr(A,row,col) += pow(x[k],rowi+coli)*pow(y[
    k],rowj+colj) ;
            }
            for(int k=0 ; k < deg ;k++)
                *gsl_vector_ptr(rhs,row) += f[k]*pow(x[k],rowi)*pow(y[k],
    rowj) ;
        }

        /* solve the linear system of equations
        */
        gsl_linalg_HH_svx(A,rhs) ;

        /* pack the expansion coefficients into the general space
        */
        alpha.clear() ;
        for(int row=0; row < n ; row++)
            alpha.push_back(gsl_vector_get(rhs,row)) ;

        gsl_vector_free(rhs) ;
        gsl_matrix_free(A) ;
#else /* HAVE_GSL */

/* the macro maps a pair of Fortran77 0-based coordinates to a
* 0-based 1D C index.
*/
#define POWFIT2D_F77_TO_C(row,col,N) ((col)*(N)+(row))
        /* the n by n matrix of the linear system of equations */
        double *A=new double[n*n] ;

        /* the RHS of the linear system of equations */
        double *rhs=new double[n] ;

        /* fill the matrix and the right hand side
        */
        for(int row=0; row<n ;row++)
        {
            int rowi, rowj ;
            strIndxInv(row,rowi,rowj) ;
            for(int col=0; col< n ;col++)
            {
                int coli, colj ;
                strIndxInv(col,coli,colj) ;
                /* debugging
                * cout << "# mapped " << row << " " << rowi << " " << rowj <<
     endl ;
                */
                A[POWFIT2D_F77_TO_C(row,col,n)] =0. ;
                for(int k=0 ; k < deg ;k++)
                    A[POWFIT2D_F77_TO_C(row,col,n)] += pow(x[
    k],rowi+coli)*pow(y[k],rowj+colj) ;
            }
            rhs[row]=0. ;
            for(int k=0 ; k < deg ;k++)
                rhs[row] += f[k]*pow(x[k],rowi)*pow(y[k],rowj) ;
        }

        /* solve the linear system of equations
        */

        integer nrhs=1 ;
        integer lda= n ;
        integer *ipiv = new integer[n] ;
        integer info = 0 ;
        dgesv_(&lda,&nrhs,A,&lda,ipiv,rhs,&lda,&info) ;
        if ( info )
            cerr << __FILE__ << " " << __LINE__ << " dgesv(1) info= " << info <
    < endl ;
        delete [] ipiv ;
```

```
        /* pack the expansion coefficients into the general space
        */
        alpha.clear() ;
        for(int row=0; row < n ; row++)
            alpha.push_back(rhs[row]) ;

        delete [] rhs ;
        delete [] A ;
#undef POWFIT2D_F77_TO_C
#endif /* HAVE_GSL */
    }
```

5.1.3.5 double PowFit2D::fitted (const double *x*, const double *y*) [inline], [private]

Calculate a value of the fitting function.

Parameters

in	*x*	Cartesian x coordinate of the 2D point
in	*y*	Cartesian y coordinate of the 2D point

Returns

the mixed polynomial (the fit) at the 2D point.

Since

2007-07-12

```
{
    double f=0. ;
    for(int i=0; i < alpha.size() ; i++)
    {
        int powx, powy ;
        strIndxInv(i,powx,powy) ;
        f += alpha[i]*pow(x,powx)*pow(y,powy) ;
    }
    return f ;
}
```

5.1.3.6 static bool PowFit2D::isComment (const string *inpli*, const regex_t & *preg*) [inline], [static], [private]

classify the input line as comment or non-comment.

```
{
    /* debugging
    * cout << __FILE__ << " " << __LINE__ << " " << inpli << endl ;
    */
    if ( regexec(&preg,inpli.c_str(),0,0,0) == 0)
        return true ;
    else
        return false ;
}
```

5.1.3.7 void PowFit2D::readf (char * *fname*, bool *verb* = false) [inline], [private]

Read the (x,y,f) triples from an ASCII file. The syntax of the lines in the ASCII file is as follows:

- Lines that start with optional white space followed by a hash (#) or containing only white space are ignored (a.k.a. comment lines)
- All other lines start with three floating point numbers separated by white space. Letters and numbers in the rest of these lines are ignored. The three floating point numbers are the x Cartesian coordinate, then the y Cartesian coordinate, then the function value f. The count of lines of this form implies the number of input triples.

Parameters

in	*fname*	the UNIX file name
in	*verb*	if true turns on verbose commenting on the inputs

Since

2007-07-12

```
{
    ifstream inf(fname) ;
    /* complain if the file is not readable.
    */
    if ( !inf)
    {
        cerr << "Cannot open " << fname << endl ;
        exit(EXIT_FAILURE) ;
    }



    /* a regular expression characterizing comment lines
    */
    regex_t preg;
    if ( int rege = regcomp(&preg,"(^[[:space:]]*#)|(^[[:space:]]*$)",
  REG_NOSUB|REG_EXTENDED) )
    {
        char errbuf[257] ;
        regerror(rege,&preg,errbuf,257) ;
        cerr << __FILE__ << " " << __LINE__ << " " << errbuf << endl ;
    }

    /* read the file line by line until EOF.
    */
    while( !inf.eof() )
    {
        /* one line of the input file
        */
        string inpli ;
        getline(inf,inpli) ;
        if ( !isComment(inpli,preg) )
        {
            double vals[3] ;
            istringstream s(inpli) ;
            s >> vals[0] >> vals[1] >> vals[2] ;
            if ( verb)
                cout << "# " << x.size() << " " << vals[0] << " " << vals[
  1] << " " << vals[2] << endl ;
            x.push_back(vals[0]) ;
            y.push_back(vals[1]) ;
            f.push_back(vals[2]) ;
        }
    }
    regfree(&preg) ;
}
```

5.1.3.8 int PowFit2D::strIndx (const int *i*, const int *j*) [inline],[private]

Derive a straightened single index from a pair index. The result is constructed by reading the infinite 2x2 array of integer indices in the first quadrant along anti-diagonals.

Parameters

in	*i*	first index of the pair, >=0
in	*j*	second index of the pair, >=0

Returns

a unique number larger or equal to zero. (0,0) is mapped on 0, (0,1) on 1, (1,0) on 2, (0,2) on 3, (1,1) on 4, (2,0) on 5 etc. So a group of common i+j produces a sequential list of integers, and the smallest i are maped on the lower single indices.

Since

2007-07-12

```
{
    return triang(i+j)+i ;
}
```

5.1.3.9 void PowFit2D::strIndxInv (const int *n*, int & *i*, int & *j*) [inline], [private]

Decompose a straightened single index into a pair of indices. This is the inverse function of strIndx().

Parameters

in	*n*	the straightened single index.
out	*i*	first index of the pair, >=0
out	*j*	second index of the pair, >=0

Since

2007-07-12

```
{
    /* the triangular number that starts a sequence of  common sum i+j
    */
    int p=0 ;
    while ( triang(p) <= n)
        p++ ;
    p-- ;
    i = n-triang(p) ;
    j=p-i ;
}
```

5.1.3.10 int PowFit2D::triang (const int *i*) [inline], [private]

Compute triangular number. OEIS A000217 [**?**]

Parameters

in	*i*	

Returns

the i'th triangular number, i(i+1)/2.

Since

2007-07-12

```
{
    return i*(i+1)/2 ;
}
```

5.1.4 Field Documentation

5.1.4.1 vector<double> PowFit2D::alpha [private]

list of expansion coefficients, 1D version (straightened)

5.1.4.2 vector<double> PowFit2D::f [private]

list of function values, one per (x,y) pair.

5.1.4.3 int PowFit2D::maxHybrP [private]

the maximum combined power of the x and y coordinates for the fit

5.1.4.4 bool PowFit2D::verb [private]

verbosity rised if true.

5.1.4.5 vector<double> PowFit2D::x [private]

list of x coordinates

5.1.4.6 vector<double> PowFit2D::y [private]

list of y coordinates, one per x coordinate

6 File Documentation

6.1 PowFit2D.cxx File Reference

Data Structures

- class PowFit2D

Macros

- #define POWFIT2D_F77_TO_C(row, col, N) ((col)*(N)+(row))

Functions

- void usage (char *argv0)
- int main (int argc, char *argv[])

6.1.1 Macro Definition Documentation

6.1.1.1 #define POWFIT2D_F77_TO_C(*row, col, N*) ((col)*(N)+(row))

6.1.2 Function Documentation

6.1.2.1 int main (int *argc,* char * *argv[]*)

The main executable.

Parameters

in	*-v*	the option `-v` can be used to produce more verbose output
in	*-p*	the option `-p` followed by a non-negative integer indicates that the maximum (hybrid) degree of the powers in the fit should be reduced to this integer. If not used, the program will calculate a default, which is to take the maximum fitting degree which still matches a least-squares algorithm. The default means that the degree is 0 for 1 Cartesian input point, the degree is 1 for 2 or 3 input points, the degree is 2 for 4 to 6 input points etc. This can be seen as always including a full set of azimuthal parameters m for each $\sim r^p \cos(m\varphi)$ and $\sim r^p \sin(m\varphi)$, $0 \leq m \leq p$, such that no directional preference is found in the algorithm.
in	*fname*	the single command line argument is the ASCII file name with input lines as described in PowFit2D::readf().

Returns

0 if the fitting was apparently successful, -1 if wrong options were used or the input file couldn't be opened.

Since

2007-07-12

```
{
    /* command line option.
    */
    int c ;

    /* Maximum radial power in the fit. Initialized with negative value
    * means we automate the calculation.
    */
    int maxP = -1 ;

    /* verbosity
    */
    bool verb=false ;

    /* read command line options
    */
    while( (c=getopt(argc,argv,"vp:")) != -1 )
        {
                switch(c)
                {
                case 'v':
            verb=true ;
            break ;
                case 'p':
            maxP = atoi(optarg) ;
            break ;
                case '?':
                        usage(argv[0]) ;
                        return -1 ;
                        break ;
                }
        }

    if ( optind >= argc)
    {
        usage(argv[0]) ;
        cerr << "Missing file name argument\n" ;
        return -1 ;
    }

    if (verb)
    {
        time_t now=time(NULL) ;
        cout << "# " << getenv("USER") << " " << ctime(&now) << "# " ;
        for(c=0 ; c < argc; c++)
            cout << argv[c] << " " ;
        cout << endl ;
    }

    /* read the 2D points to be fitted from the ASCII File
    */
    PowFit2D fitProbl(argv[optind],verb) ;

    /* do the fitting.
    */
    fitProbl.fit(maxP) ;

    /* show the results
    */
    fitProbl.display(verb) ;

    return 0 ;
}
```

6.1.2.2 void usage (char * *argv0*)

Print a usage information.

Parameters

in	*argv0*	the command name

Since

2007-07-12

```
{
        cout << "usage : " << argv0 << " [-v] [-p maxdegree] inptfile\n" ;
}
```

Index