Linear Least-Squares Data Fitting with Orthogonal Polynomials

G.J. Fee
Centre for Experimental and Constructive Mathematics,
Department of Mathematics
Simon Fraser University, Burnaby, Canada, V5A 1S6
gjfee@cecm.sfu.ca

1 Abstract

We consider the problem of fitting data to a polynomial. We review Maple's least-squares methods, which include the normal equations method (NE), the orthogonal factorization method (QR), and the singular value decomposition method (SVD), which are available in the following Maple packages: `stats`, `linalg` and `LinearAlgebra`. We implement three orthogonal polynomial methods in this paper and compare them with the other methods.

The inputs to our weighted linear least-squares problem are as follows: N a positive integer which is the number of data points, X a real vector with N components $X_1, X_2, \ldots X_N$ which define the N values of the independent variable called x, Y a real vector with N components $Y_1, Y_2, \ldots Y_N$ which define the N corresponding values of the dependent variable called y, S a real vector with N positive components $S_1, S_2, \ldots S_N$ which contains the N corresponding estimated standard deviations of each of the N dependent variables. It is more common to be given a vector W of N weights, but we will calculate the vector of weights by the formula $W_i = S_i^{-2}$ for i from 1 to N. Notice that we have assumed that the independent vector values $X_1, X_2, \ldots X_N$ are exact, so we don't need a vector containing the estimated standard deviations in the values of the independent variable. In the general case of the weighted linear least-squares problem, we require some basis functions so we can express the output as a linear combination of the given basis functions. As we are concerned with polynomial fitting, we just require n the maximum degree of the polynomial fit. We assume $n < N$.

Given any real-valued function f(x) that is defined at each of our data points X_i for i from 1 to N. We define the error vector E by $E_i = Y_i - f(X_i)$ for i from 1 to N. If $f(x)$ also depends on some parameters then the weighted least-squares algorithm chooses the parameters such that the weighted sum of squares of the errors $\sum_{i=1}^{N} W_i \left(Y_i - f(X_i)\right)^2$ is minimized. In general, this leads to a system of non-linear equations to determine the parameters. If the parameters enter

linearly, such as the undetermined coefficients in a linear combination of basis functions, then the weighted least-squares approach leads to a linear system of equations to find the parameters. If we measure each error E_i in terms of the number of standard deviations then we just divide E_i by S_i, and the square of this leads to the formula for the weights given above. We think it would be easier for an experimentist to estimate the standard deviation in a measurement than to estimate the weight that should be used in a weighted least-squares problem, so our input includes the vector of estimated standard deviations, rather than the vector of weights. In the case of given weights, then the solution of the i^{th} weight equation for the i^{th} standard deviation is $S_i = W_i^{-1/2}$.

Example 1: There are 201 equally spaced values on the interval $[0, 1]$ for the vector of independent variable values, that is $X[i] = (i-1)/200$ for i from 1 to 201. All estimated standard deviations are 1, so all weights are also 1 and we have a conventional unweighted least-squares problem. There are 201 simulated measurements, that is $Y_i = f(X_i) + G_i$ for i from 1 to 201, where G_i is the value of a random variable selected from a standard normal distribution with mean zero and standard deviation one. The chosen function is $f(x) = 10^5 \exp(-x) \sin(1 + 41\,x + 29\,x^2)$. The problem is to fit a degree 40 polynomial to this data, where the fit is in the least-squares sense. We use the standard monomial basis functions so that $b_i(x) = x^{i-1}$ for i from 1 to 41.

This leads to a design matrix A with 201 rows and 41 columns. The symmetric matrix $A^t A$ has an infinity norm condition number equal to $0.9e61 = 0.9 \times 10^{61}$. Using standard double precision arithmetic, all the above methods produce unacceptable results.

We can always increase the precision to overcome the ill-conditioning, but a better approach is to use an orthogonal polynomial basis instead of using the standard monomial basis functions. Our first method uses the shifted Legendre polynomials as basis functions. Our second method uses the discrete Chebyshev polynomials. Both of these orthogonal polynomials are defined in the book *Handbook of Mathematical Functions*, by M. Abramowitz and A. Stegun.

The discrete Chebyshev polynomials have been generalized to the case of arbitrary positive weights and arbitrary independent variable values. Our third method is a Maple implementation of the algorithm in the book *Chebyshev Polynomials in Numerical Analysis* by L. Fox and I.B. Parker. The orthogonal polynomial methods produce acceptable results, using double precision arithmetic.

2 The standard solution methods

There are three standard solution methods: the normal equations (NE) method, the QR decomposition method, and the singular value decomposition (SVD) method. They are listed in the order of the numerical quality of the result. The normal equations method squares the condition number of the problem, so suffers the most roundoff error, but it is the fastest method. The SVD method gives the best numerical results, but it is the slowest. The QR decomposition method is between the other two in both numerical quality and execution speed. The above three methods can handle any basis functions, not just polynomials.

We review the normal equations method. Choose m linearly independent basis functions $b_1(x), b_2(x), \ldots b_m(x)$. For the case of a polynomial fit we could choose the standard monomial basis functions $b_j(x) = x^{j-1}$. Create the function $f(x) = \sum_{j=1}^{M} c_j b_j(x)$ as a linear combination of the basis functions, where $c_1, c_2, \ldots c_m$ are the m parameters. Evaluate the function $f(x)$ at each of the N data points to compute the error vector E. Scale each error component by dividing it by its corresponding estimated standard deviation. Then we form the sum of the squares of the scaled errors. A necessary condition for a minimum of this sum is that all the first partial derivatives with respect to the parameters must be set equal to zero. This generates the following symmetric linear system of equations: $M^t D^t D M c = M^t D^t D Y$ where Y is the vector of dependent variable observations, D is the diagonal matrix which contains the reciprocals of the corresponding estimated standard deviations along the diagonal, M is the N by m matrix such that $M_{i,j} = b_j(X_i)$, M^t denotes the transpose of the matrix M, $D^t = D$ since D is a diagonal matrix. and c is the vector of unknown parameters. These equations are called the normal equations. They can be solved by gaussian elimination, by performing an LU decomposition of the coefficient matrix, or since the coefficient matrix is positive definite, we can solve the symmetric linear system, by doing a Cholesky decomposition of the coefficient matrix, that is express it as LL^t where L is a lower triangular matrix, and then solve for the vector c.

The QR algorithm first forms the matrix $A = DM$ and the scaled vector DY. We see that we now have a conventional unweighted least-squares problem with coefficient matrix A and right hand side vector DY. The QR algorithm first expresses $A = QR$ where Q has orthonormal columns and R is upper triangular. This can be obtained by the Gram-Schmidt algorithm, or better by the modified Gram-Schmidt algorithm or best by Householder reflectors. Then we form the vector $Q^t DY$ and solve the upper trianguler linear system with coefficient matrix R and the newly created vector on the right hand side, for the unknown parameters.

The singular value decomposition first expresses the above matrix A as $A = USV^t$ where U is an orthogonal matrix or a rectangular matrix with orthonormal columns, S is a diagonal rectangular matrix with the singular values of A along the diagonal in descending order, and V^t is an orthogonal matrix. The singular value decomposition can be computed by using Householder reflectors to reduce A to bidiagonal form, then a modified eigenvalue QR algorithm can used reduce the size of the off diagonal elements so that they are all less than the given off diagonal tolerance. The solution vector is $VS^{-1}U^t DY$ where S^{-1} is obtained from S by replacing the large diagonal elements by their reciprocals, where large means any number that is greater than the given singular value tolerance.

All the above methods can be implemented so that they have a computational complexity of $O(Nm^2)$ floating point operations and a memory complexity of $O(Nm)$ floating point numbers.

3 Maple's least-squares procedures

Maple [1] contains several least-squares procedures, such as,
`stats[fit,leastsquare]`,
`linalg[leastsqrs]`,
`LinearAlgebra[LeastSquares]` and
`CurveFitting[LeastSquares]`.
Maple also has the QR and SVD algorithms in
`evalf(Svd())`,
`linalg[QRdecomp]`,
`LinearAlgebra[QRDecomposition]` and
`LinearAlgebra[SingularValues]`.

The normal equations method is used in both the `stats` and `linalg` packages. The QR algorithm is the default algorithm used in the `LinearAlgebra` package. The `CurveFitting` package just calls the `LinearAlgebra` package, but it does allow for weights.

We ran several least-squares procedures at a few settings of `Digits`. The `time` column is CPU time measured in seconds with Maple's `time()` procedure. The `error` column is the 2-norm of the difference between the computed solution at the given setting of `Digits` and the computed solution at `Digits:=128;`. The `cond(A,2)` column is the 2-norm condition number of the design matrix A, which is the largest singular value of A divided by the smallest singular value of A. The `cond(`A^tA`,2)` column is the 2-norm condition number for the matrix product of the transpose of A times A. The condition number columns would be the same for all algorithms, since the same design matrix A was common input to all of the procedures. All tests were run on a DEC ALPHA computer, using a 64-bit version of Maple 7. Here are some timings for Example 1.

`stats[fit,leastsquare]`

Digits	time	error
16	9.672	4.32332e+04
32	19.980	3.01829e+04
48	38.445	1.06767e+04
64	62.935	3.52278e-03
80	100.505	6.72389e-19

`linalg[leastsqrs]`

Digits	time	error	cond(A,2)	cond(A^tA,2)
16	6.348	4.19421e+04	2.24853e+18	1.12199e+18
32	8.263	2.98281e+04	1.76026e+30	1.61285e+34
48	9.830	1.43902e+04	1.76022e+30	5.24675e+49
64	10.435	2.28023e-02	1.76022e+30	3.09846e+60
80	10.562	2.18639e-19	1.76022e+30	3.09837e+60

`linalg[QRdecomp]`

Digits	time	error
16	167.526	5.21543e+04
32	257.615	1.68712e+00
48	363.538	2.41451e-16

`evalf(Svd())`

Digits	time	error
16	172.816	3.30814e+04
32	267.279	2.29236e+01
48	352.360	1.62756e-15

`LinearAlgebra[` normal equations `]`

Digits	time	error
16	1.950	4.87161e+04
32	2.363	2.95537e+04
48	3.134	1.34027e+04
64	3.761	1.00734e-02
80	5.328	3.90552e-19

`LinearAlgebra[LeastSquares]`

Digits	time	error
16	2.144	3.84802e+04
32	2.803	1.57575e+01
48	3.735	1.47829e-15
64	6.467	2.12608e-31
80	7.977	1.26784e-47

`LinearAlgebra[SingularValues]`

Digits	time	error
16	28.193	3.30826e+04
32	44.920	9.45847e+00
48	132.777	1.78400e-15
64	236.261	1.02603e-31

4 Least-Squares using Legendre Polynomials

The ill-conditioning in the natural-basis least-squares methods can be cured by using an orthogonal polynomial basis. The Legendre polynomials form an orthogonal system of polynomials with respect to the inner product defined by

$$< f, g >= \int_{-1}^{1} f(x)\, g(x)\, dx$$

where f and g are Lebesque measurable real valued functions defined on the real interval $[-1, 1]$. The Legendre polynomials [2] can be defined according to the following recurrence relation:

$$P_0(x) = 1, P_1(x) = x,$$

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x).$$

We say functions f and g are orthogonal if the inner product $< f, g >= 0$. The Legendre polynomials form an orthogonal system of polynomials, that is

$$\int_{-1}^{1} P_m(x)\, P_n(x)\, dx = 0$$

for $0 <= m < n$ where m and n are distinct non-negative integers. The Legendre polynomials are not orthonormal. The normalization condition is

$$\int_{-1}^{1} (P_n(x))^2\, dx = 2\,(2\,n+1)^{-1}$$

They also satisfy the equation $P_n(1) = 1$ for all non-negative integers n.

The linear transformation $L(x) = (2x - b - a)/(b - a)$ of the argument to a Legendre polynomial translates the interval of orthogonality from $[-1, 1]$ to $[a, b]$. For example the shifted Legendre polynomials $P_n^*(x) = P_n(2x - 1)$ are orthogonal on the interval $[0, 1]$. In general the set of translated Legendre polynomials $P_n((2x - b - a)/(b - a))$ form an orthogonal system on the interval $[a, b]$.

We now use the normal equations approach to solve Example 1, by using the shifted Legendre polynomials as our basis functions instead of the standard monomials. We use the above recurrence relation to compute the values of the Legendre polynomials at any given argument. The `hf` under the `Digits` column denotes the use of hardware floating point arithmetic using Maple's `evalhf()` procedure. The results are:

normal equations using shifted Legendre polynomials

Digits	time	error	cond(A,2)	cond(A^tA,2)
hf	.062	7.81493e-11	1.18022e+01	1.39292e+02
16	2.632	1.37536e-10	1.18022e+01	1.39292e+02
32	3.287	2.11346e-26	1.18022e+01	1.39292e+02

5 Least-Squares using Chebyshev Polynomials

We now use the Chebyshev polynomials of the first kind instead of the Legendre polynomials. The Chebyshev polynomials form an orthogonal system of polynomials with respect to the weighted inner product defined by

$$< f, w, g >= \int_{-1}^{1} \frac{f(x)\, g(x)}{\sqrt{1-x^2}} dx$$

where w is the weight function $(1 - x^2)^{-1/2}$. The Chebyshev polynomials [2] can be defined according to the following recurrence relation:

$$T_0(x) = 1, T_1(x) = x,$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

The normalization conditions are $< T_0, w, T_0 >= \pi$ and $< T_j, w, T_j >= \frac{\pi}{2}$, for positive integer j. They also satisfy the equation $T_n(1) = 1$ for all non-negative integers n.

We now use the normal equations approach to solve Example 1, by using the shifted Chebyshev polynomials as our basis functions instead of the standard monomials. This algorithm can be found in [3] in chapter 12 section 2. The results are:

normal equations using shifted Chebyshev polynomials

Digits	time	error	cond(A,2)	cond(A^tA,2)
hf	.045	7.91818e-11	6.33780e+00	4.01677e+01
16	2.170	1.25604e-10	6.33780e+00	4.01677e+01
32	2.813	1.21656e-26	6.33780e+00	4.01677e+01

6 The Discrete Chebyshev polynomial solution

We have seen that the classical orthogonal polynomials can overcome the ill-conditioning in the least-squares problem, but an even better approach is to use discrete orthogonal polynomials [4] instead of using the standard monomial or classical orthogonal polynomial basis functions. If we have a standard least-squares problem with all weights unity and $X_i = i - 1$ for i from 1 to N, then the appropriate orthogonal polynomials are the discrete Chebyshev [2] polynomials. They have been generalized [5] to the case of arbitrary positive weights and arbitrary independent variable values. The solution to the weighted least-squares problem is $q(x)$ where

$$q(x) = \sum_{x=0}^{n} c_i p_i(x)$$

in which $p_i(x)$ is the degree i, monic generalized discrete Chebyshev polynomial. We define a discrete weighted inner product $\langle f(x), g(x) \rangle$ by

$$\langle f(x), g(x) \rangle = \sum_{j=1}^{N} W_j f(X_j) g(X_j)$$

then the $p_i(x)$ form an orthogonal system of polynomials with respect to the above inner product. This means that $\langle p_i(x), p_j(x) \rangle = 0$ for $0 \le i < j \le n$. The coefficients in the orthogonal polynomial expansion are given by

$$c_i = \frac{\langle Y, p_i(x) \rangle}{\langle p_i(x), p_i(x) \rangle}$$

in which Y is the vector of dependent variable values and we extend the definition of the inner product so that

$$\langle Y, p_i(x) \rangle = \sum_{j=1}^{N} W_j Y_j p_i(X_j)$$

A practical method to compute the values of the monic generalized discrete Chebyshev polynomials is to use the following [5] three term recurrence relation.

$$p_0(x) = 1, p_1(x) = x - a_1,$$

$$P_i(x) = (x - a_i)P_{i-1}(x) - b_i P_{i-2}(x),$$

in which the recurrence relation coefficients are given by

$$a_i = \frac{\langle x p_{i-1}(x), p_{i-1}(x)\rangle}{\langle p_{i-1}(x), p_{i-1}(x)\rangle}$$

$$b_i = \frac{\langle p_{i-1}(x), p_{i-1}(x)\rangle}{\langle p_{i-2}(x), p_{i-2}(x)\rangle}$$

One of the advantages of the discrete orthogonal polynomial approach is that we can find a lower degree least-squares polynomial, by dropping the higher degree terms, whereas the other methods using the standard monomial basis functions or the classical orthogonal polynomials require a recomputation. The other methods have a computational time complexity of $O(Nn^2)$ floating point operations, whereas the discrete orthogonal polynomial approach has a time complexity of $O(Nn)$ to compute the degree n least-squares polynomial that fits N data points. If we want to compute all the polynomial fits up to degree n, then the standard approach would have a time complexity of $O(Nn^3)$, whereas the time complexity of the discrete orthogonal polynomial approach would remain at $O(Nn)$. The memory space complexity of the standard methods is $O(Nn)$ floating point numbers, but the discrete orthogonal polynomial method requires just $O(N + n)$ numbers.

The discrete orthogonal polynomial method finds all the least-squares fitted polynomials up to degree 40 for the data given in Example 1, using double precision arithmetic and produces acceptable results.

monic discrete Chebyshev polynomials

Digits	time	error
hf	.075	1.00096e-09
16	1.174	6.48027e-09
32	1.344	4.90378e-25

If we use orthonormal discrete Chebyshev polynomials instead of monic discrete Chebyshev polynomials to solve the least-squares problem, then the CPU time increases slightly as shown in the following table.

orthonormal discrete Chebyshev polynomials

Digits	time	error
hf	.089	7.26114e-10
16	1.610	2.72974e-09
32	1.859	2.21080e-25

The reason we have introduced orthonormal polynomials, was to avoid exponent overflow problems, which occur in Example 2, while using monic discrete Chebyshev polynomials.

Example 2: There are 10001 equally spaced values on the interval $[0,1]$ for the vector of independent variable values, that is $X[i] = (i-1)/10000$ for i from 1 to 10001. All estimated standard deviations are $1.0e-6$. There are 10001 simulated measurements, that is $Y_i = f(X_i) + G_i$ for i from 1 to 10001, where G_i is the value of a random variable selected from a normal distribution with mean zero and standard deviation $1.0e-6$. The chosen function is $f(x) = Ai(-100x)$, where $Ai(x)$ is Airy's Ai function, which is denoted by `AiryAi(x)` in Maple. The problem is to fit a degree 429 polynomial to this data.

The `error` column is the 2-norm of the difference between the computed least-squares polynomial and the exact solution $Ai(-100x)$. The results are:

orthonormal polynomials for Example 2

Digits	time	error
hf	48.418	7.48767e-07

References

[1] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter and J. MacCarron, *Maple 6 Programming Guide*, Waterloo Maple Inc., Waterloo, Canada (2000).

[2] M. Abramowitz and A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, New York (1972).

[3] Ward Cheney and David Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole Monterey, California (1980).

[4] D.G. Kleinbaum, L.L. Kupper and K.E. Muller, *Applied Regression Analysis and Other Multivariable Methods*, PWS-KENT Publishing, Second edition (1988).

[5] L. Fox and I.B. Parker, *Chebyshev Polynomials in Numerical Analysis* Oxford University Press (1968).