

An Algorithm for Least-Squares Polynomial Approximation

Michael D. Godfrey
Imperial College of Science and Technology
As of 1989 at: ISL, Stanford University
godfrey@isl.stanford.edu

1. Introduction

A classical analysis shows that an appropriately stated least squares polynomial approximation problem leads to a system of linear equations of the form

$$b = H_n a. \tag{1.1}$$

Where b is a $(1 \times n)$ vector with components:

$$b_j = \int_0^1 x^{j-1} f(x) dx,$$

$f(x)$ is the function to be approximated, a is the $(n \times 1)$ vector of coefficients of the polynomial of order $n - 1$, and H_n is a Hilbert matrix of order n . The fact that the solution of equation 1.1 is given by:

$$a = H_n^{-1} b \tag{1.2}$$

and H_n is ill-conditioned even for moderate n ($\text{cond}(H_8) = 1.53 \times 10^{10}$) has made this a popular example of the numerical difficulty of obtaining accurate results in polynomial fitting procedures.

This example does indicate the truth of the general proposition that polynomials are difficult to deal with numerically. However, inspection of equation 1.2 also suggests a useful procedure for accurate solution of this particular problem. While Hilbert matrices are ill-conditioned and, therefore, their direct inversion requires both care and high precision, there is, in fact, no reason to compute H_n^{-1} from H_n . The coefficients of H_n^{-1} can be derived analytically. These coefficients are integer valued for any n . Thus, H_n^{-1} can be computed exactly. This fact is the basis for the following algorithm. The only error involved in computing the algorithm is that which will arise in the numerical evaluation of

$$b_j = \int_0^1 x^{j-1} f(x) dx. \tag{1.3}$$

Evaluation of this definite integral may prove difficult due to unfortunate behavior of $f(x)$, which must be scaled to be on the interval $[0, 1]$. However, this is not inherently a difficult numerical exercise.

2 An Algorithm for Least-Squares Polynomial Approximation

2. Statement of the Algorithm

2.1 Choose a value for n .

2.2 Evaluate

$$b_j = \int_0^1 x^{j-1} f(x) dx, \quad j = 1, \dots, n$$

using whatever procedures are appropriate for $f(x)$.

Next, evaluate the expression for H' (with elements $h'_{i,j}$):

$$h'_{i,j} = (-1)^{i+j} (i+j-1) \binom{n+i-1}{n-j} \binom{n+j-1}{n-i} \binom{i+j-2}{i-1}^2$$

2.3 Compute H_n^{-1} using the *Octave* `invhilb(n)` function which uses an accurate algorithm by Dirk Laurie. (See the Octave help `invhilb` information or look up the algorithm at <http://ideas.repec.org/c/cod/octave/c090904.html>.) In outline, his algorithm is:

2.3.1 $k = 1 : n$;

$$p = k \times \binom{k+n-1}{k-1} \times \binom{n}{k};$$

$$p(2 : 2 : n) = -p(2 : 2 : n);$$

2.3.2 $h(i, :) = (p(i) \times p) ./ (i : i + n - 1); \quad i = 1, \dots, n$

This algorithm is precise through $n = 203$. The algorithm continues after $n = 203$ using an algorithm which produces the nearest representable values until double precision floating point overflow occurs.

2.4 Compute

$$a_i = \sum_{j=1}^n b_j h'_{i,j} \quad i = 1, \dots, n$$

3. Conclusion

Since recomputing for increasing values of n is relatively quick, a useful procedure is to start with a small value of n and increase this value to get improved fits. A stopping rule should be based on the behavior of

$$E(n) = \int_0^1 \left(f(x) - \sum_{i=1}^n a_i x^{i-1} \right)^2 dx$$

as a function of n .