Orthogonal Polynomial Curve Fitting

Jeff Reid

This document will explain how to implement curve fitting using orthogonal polynomial least square techniques. This process is given a set of weighted data points (w_i, x_i, y_i), and generates a function that "best" fits the data for a given maximum degree of polynomial. This generated function can be used to "smooth" the existing data points, or to calculate new points (polynomial interpolation).

The generated polynomial will be of the form:

$$g(x) = \sum_{j=0}^{m} b_j p_j(x)$$

Where b_j is a set of *m+1* constants, and p_j is a set of *m+1* orthogonal polynomials. These polynomials are designed to have two properties: a recursive definition, and orthogonality.

The recursive definition of the polynomials **(eq 1)**:

$$p_{j+1}(x) = (x - A_{j+1})\, p_j(x) - B_j\, p_{j-1}(x)$$
$$p_0(x) = 1$$
$$p_{-1}(x) = 0$$

Where *A* and *B* are sets of constants.

A set of *p* polynomials is orthogonal if **(eq 2)**:

$$\sum_{i=0}^{n-1} w_i\, p_j(x_i)\, p_k(x_i) = 0$$

$$\text{when } j \neq k$$

In other words, a polynomial of any degree (*j*) will be linearly independent of any polynomial of a different degree (*k*), or even any linear combination of polynomials of degree other than *j*.

To generate the *p* polynomials, the sets of constants for *A*, and *B* need to be determined.

Using the recursive definition of *p* for case *j=0*:

$$p_1(x) = (x - A_1)$$

combine with the orthogonal property:

$$\sum_{i=0}^{n-1} w_i p_0(x_i) p_1(x_i) = 0 = \sum_{i=0}^{n-1} w_i (x_i - A_1)$$

resulting in:

$$A_1 = \frac{\sum_{i=0}^{n-1} w_i x_i}{\sum_{i=0}^{n-1} w_i}$$

Generate *k+1* equations, (*j* goes from *0* to *k*):

$$\sum_{i=0}^{n-1} w_i p_j(x_i) p_{k+1}(x_i) = 0$$

and substitute using recursive property **(eq 3)**:

$$\sum_{i=0}^{n-1} w_i x_i p_j p_k - A_{k+1} \sum_{i=0}^{n-1} w_i p_j p_k - B_k \sum_{i=0}^{n-1} w_i p_j p_{k-1} = 0$$

For the cases where *j* goes from *0* to *k-2*, all three terms are zero due to the orthogonal property. (In the first term on the left, $x_i p_j$, can be expressed as a combination of polynomials from p_0 to p_{k-1} which is why it is zero).

For *j* = *k-1* the middle term of **(eq 3)** is still zero, leading to:

$$B_k = \frac{\sum_{i=0}^{n-1} w_i x_i p_{k-1} p_k}{\sum_{i=0}^{n-1} w_i (p_{k-1})^2}$$

using the recursive property to replace $x_i p_{k-1}$:

$$p_k(p_k) = p_k(x_i p_{k-1} - A_k p_{k-1} - B_{k-1} p_{k-2})$$

the *A* and *B* terms are zero (orthogonal property) leading to:

$$B_k = \frac{\sum_{i=0}^{n-1} w_i (p_k)^2}{\sum_{i=0}^{n-1} w_i (p_{k-1})^2}$$

For *j* = *k* the rightmost term of **(eq 3)** is zero, leading to:

$$A_{k+1} = \frac{\sum_{i=0}^{n-1} w_i x_i (p_k)^2}{\sum_{i=0}^{n-1} w_i (p_k)^2}$$

Using the above equations, and given a set of weighted data points, (w_i, x_i), both *A* and *B* constant sets can be generated.

The definition for the function to be generated by least squares technique is:

$$g_m(x) = \sum_{j=0}^{m} b_j p_j(x)$$

where *b* is a set of constants to be determined. Define *F* to be sum of the squares of deviations:

$$F(b) = \sum_{i=0}^{n-1} w_i (y_i - g_m(x_i))^2$$

substitute for g_{m}:

$$F(b) = \sum_{i=0}^{n-1} w_i (y_i - \sum_{j=0}^{m} b_j p_j(x_i))^2$$

Least squares technique involves generating *m+1* partial derivatives of *F* with respect to each of the *b* constants, and determining a set of *b*'s that result in all partial derivatives = 0, minimizing the sum of deviations. The *m+1* partial derivative equations to be solved are (*k* goes from *0* to *m*):

$$\frac{\partial F}{\partial b_k} = -\,2\sum_{i=0}^{n-1} w_i (y_i - \sum_{j=0}^{m} b_j p_j(x_i)) p_k(x_i) = 0$$

This leads to:

$$\sum_{j=0}^{m} \sum_{i=0}^{n-1} (w_i p_j(x_i) p_k(x_i)) b_j = \sum_{i=0}^{n-1} w_i y_i p_k(x_i)$$

Each of these *m+1* equations contains all *m+1 b* constants, and a technique of simultaneously solving the *m+1* the equations to determine the *b*'s could be used (such as matrix inversion). However, note that left sides of the equations are zero except for the cases *j*=*k*, due to orthogonality (eq 2). This allows the *b*'s to be calculated directly:

$$b_k = \frac{\sum_{i=0}^{n-1} w_i y_i p_k(x_i)}{\sum_{i=0}^{n-1} w_i (p_k(x_i))^2}$$

This system, given a set of weighted data points (wi, xi, yi), can be used to generate the required sets of constants, *A*, *B*, and *b*.

Algorithm

```
input:  w[n], x[n], y[n];

output: b[m+1], A[m+1], B[m+1], L[m+1], W[m+1],
        p[m+2][n];

    A[0] = 0.;
    B[0] = 0.;
    L[0] = 0.;
    for(i = 0; i < n; i++){
        p[0][i] = 1.;
        p[-1][i] = 0.;
        L[0] += w[i];}
    for(j = 0; 1; j++){
        W[j] = 0.;
        for(i = 0; i < n; i++){
            W[j] += w[i]*y[i]*p[j][i];}
        b[j] = W[j]/L[j];
        if(j == m)
            break;
        A[j+1] = 0.;
        for(i = 0; i < n; i++){
            A[j+1] += w[i]*x[i]*p[j][i]*p[j][i]/L[j];}
        for(i = 0; i < n; i++){
            p[j+1][i] = (x[i]-A[j+1])*p[j][i]-B[j]*p[j-1][i];}
        L[j+1] = 0.;
        for(i = 0; i < n; i++){
            L[j+1] += w[i]*p[j+1][i]*p[j+1][i];}
        B[j+1] = L[j+1]/L[j];}
```

Calculate g(x)

```
input:  x, b[m], A[m], B[m];

output: g;
    if(m == 0){
         g = b[0];
         return;}
    if(m == 1){
        g = b[0]+(x-A[1])*b[1];
        return;}
    q1 = b[m];
    q0 = b[m-1]+(x-A[m])*q1;
    for(i = m-2; i >= 0; i--){
        q2 = q1;
        q1 = q0;
        q0 = b[i]+(x-A[i+1])*q1-B[i+1]*q2;}
    g = q0;
```

Generate Standard Coefficients

This algorithm will convert the three sets of constants, *b*, *A*, and *B*, into a set of power series coefficients *c* for the generator polynomial:

$$g(x) = c_0 + c_1 x + c_2 x^2 + ... + c_m x^m$$

```
input:  b[m+1], A[m+1], B[m+1];

uses:   p0[m+1], p1[m+1], p2[m+1];

output: c[m+1];

    for(i = 0; i <= m; i++){
        c[i] = p2[i] = p1[i] = p0[i] = 0.;}

    p0[0] = 1.;
    c[0] += b[0]*p0[0];

    for(j = 1; j <= m; j++){
        p2[0] = p1[0];
        p1[0] = p0[0];
        p0[0] = -A[j]*p1[0]-B[j-1]*p2[0];
        c[0] += b[j]*p0[0];
        for(i = 1; i <= j; i++){
            p2[i] = p1[i];
            p1[i] = p0[i];
            p0[i] = p1[i-1]-A[j]*p1[i]-B[j-1]*p2[i];
            c[i] += b[j]*p0[i];}}
```