Multivariate Interpolation

By: Jason Balog and Shayne Linhart

Abstract

Demonstrate multivariate interpolation using the Vandermonde matrix method. We will be designing an algorithm that creates a polynomial interpolation given some sample data. This algorithm will be analyzed to determine where it is most useful. Specifically, we will be looking at different types of functions and sample sizes.

Introduction

Interpolation is a method used to approximate a function by using a sample data set. Multivariate interpolation involves interpolating a function of more than one variable. In many scientific applications there will be functions of multiple parameters required for computation. Some examples of these functions are the weather (variables: temperature, wind speed, humidity), people (variables: height, weight, hair color), and gas mileage (variables: speed, tire pressure, wind speed). Therefore, having a way to interpolate samples of these studies allows researchers to predict future outcomes like forecasting the weather or calculating the cost of gas for a road trip.

There are many methods used to do multidimensional interpolation which each have their own strengths and weaknesses. The purpose of this paper is to examine a specific algorithm used for multivariate interpolation and report on its performance and where it can be improved. Three-dimensional graphs will be used to visually display the resulting interpolated polynomials. This along with error analysis will be our means of judging the success of our interpolated polynomial.

Related Works

(to be written)

Algorithm

The Vandermonde matrix method involves creating an *n x n* matrix where each row is a linear combination of bivariate monomials which span a set of *basis functions*. The basis functions are created given n data points $((x_0, y_0), z_0), ((x_1, y_1), z_1), \ldots, ((x_{n-1}, y_{n-1}), z_{n-1})$, such that the degree of the polynomial m is determined by the equation below:

$$m = floor(\sqrt{(n-1)}).$$

An example of the polynomial created for 9 inputs, which is of degree 2, is as follows:

$$z(x,y) = c_8 x^2 y^2 + c_7 x y^2 + c_6 x^2 y + c_5 y^2 + c_4 x^2 + c_3 xy + c_2 x + c_1 y + c_0$$

Furthermore, polynomials of higher degree follow the same pattern. The coefficients are determined by solving the equation V*c = z using Gaussian elimination, where V is the Vandermonde matrix, *c* are the coefficients, and *z* is the solution vector given in the data set. The Vandermonde matrix spans the basis function creating a unique bivariate polynomial of the form above for each point $((x_0, y_0), z_0), ((x_1, y_1), z_1), \ldots, ((x_{n-1}, y_{n-1}), z_{n-1})$.

$$\begin{pmatrix} x_0^2y_0^2 & x_0y_0^2 & x_0^2y_0 & y_0^2 & x_0^2 & x_0y_0 & y_0 & x_0 & 1 \\ x_1^2y_1^2 & x_1y_1^2 & x_1^2y_1 & y_1^2 & x_1^2 & x_1y_1 & y_1 & x_1 & 1 \\ x_2^2y_2^2 & x_2y_2^2 & x_2^2y_2 & y_2^2 & x_2^2 & x_2y_2 & y_2 & x_2 & 1 \\ x_3^2y_3^2 & x_3y_3^2 & x_3^2y_3 & y_3^2 & x_3^2 & x_3y_3 & y_3 & x_3 & 1 \\ x_4^2y_4^2 & x_4y_4^2 & x_4^2y_4 & y_4^2 & x_4^2 & x_4y_4 & y_4 & x_4 & 1 \\ x_5^2y_5^2 & x_5y_5^2 & x_5^2y_5 & y_5^2 & x_5^2 & x_5y_5 & y_5 & x_5 & 1 \\ x_6^2y_6^2 & x_6y_6^2 & x_6^2y_6 & y_6^2 & x_6^2 & x_6y_6 & y_6 & x_6 & 1 \\ x_7^2y_7^2 & x_7y_7^2 & x_7^2y_7 & y_7^2 & x_7^2 & x_7y_7 & y_7 & x_7 & 1 \\ x_8^2y_8^2 & x_8y_8^2 & x_8^2y_8 & y_8^2 & x_8^2 & x_8y_8 & y_8 & x_8 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{pmatrix} = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \end{pmatrix}$$

Figure 1

System of equations, 9 inputs of degree 2

The algorithm must be able to create this matrix given a set of data points, determine the degree of the polynomial, create the basis function, and fill in the matrix row by row using the same index of x and y values for each row. Then the algorithm solves the system of equations for the coefficients and creates a general polynomial which can be used to interpolate any set of data points.

```
%%% Highest level function that returns all the 3D values.
%%% xi: x-inputs
%%% yi: y-inputs
%%% zi: z-inputs
%%% X,Y: together form the X-Y plane
function interp  = Interpolate(xi, yi, zi, X, Y)
            linspaceSize = 100; % Length of linspace.

            if(xi.length ~= yi.length || xi.length ~= zi.length) % Check for valid input
                interp = NaN; disp('Sample Inputs are not the same size');
                return;
            end
            m = length(xi); % m represents the number of data inputs
            Vand = createVandermonde(m, xi, yi); % Create the Vandermonde matrix
            coef = Vand\zi(:); % Find the coefficient matrix by using Gauss operator.

            % Foreach discrete point on the XY plane
            for i=1:linspaceSize
                for j=1:linspaceSize
                polyAtPoint = createCombination(m, X(i, j), Y(i, j)); % Create the polynomial combination (Same ones used in Vandermonde Matrix)
                interp(i, j) = polyAtPoint * coef; % multiply the coefficient matrix by the respecitive polynomial to get the associated z value.
                end
            end
end
```

```
%%% High level function that controls the creation of the Vandermonde matrix
function V = createVandermonde(m, xi, yi)
   V = zeros(m, m);

   for row=1:m
       sprintf('xi(row) = %20.16f', xi(row));
       V(row,:) = createCombination(m, xi(row), yi(row)); % Lower level function that creates a row of the Vandermonde Matrix
   end
end
```

```
%%% Creates a polynomial combination (rows of the Vandermonde matrix)
function c = createCombination(m, x, y)
   maxDegree = floor(sqrt(m - 1)); % Forumla for maximum degree of the polynomial combination
   tempIndex = 1; % Used to index temporary vector of terms

   % Create two vectors
   % temp1 will contain terms where x is higher degree than y (or same)
   % temp2 will contain terms where x is of lower degree than y (temp2 will
   % contain NaNs when the degrees are equal
   for i=maxDegree:-1:0
      for j=i:-1:0
          temp1(tempIndex) = (x^i) * (y^j);
          if(i ~= j)
            temp2(tempIndex) = (x^j) * (y^i);
          else
              temp2(tempIndex) = NaN;
          end
          tempIndex = tempIndex + 1;
      end
   end

   % Weave the two temporary function so that the resulting vector will
   % contain only as many terms that are needed, and ordered in a efficient
   % way (using as few higher degree terms as possible)
   c = weaveFunctions(temp1, temp2, m);
end
```

```
%Takes two vectors and weaves them together, resulting vector is
%of length, size. Starts with first argument, skips and NaNs.
function w = weaveFunctions(vector1, vector2, size)
    p1 = length(vector1); p2 = length(vector2);
    p_switch = 1;
    w = [];
    i = size;

    % Alternate elements from temp1 and temp2 into w, skip NaNs
    while(i > 0)
        if(p_switch == 1) % If its temp1's turn in the alternation
            if(isnan(vector1(p1)))
               i = i + 1; % Since NaN is not used, move w's index back
            else
               w(i) = vector1(p1);
           end
           p1 = p1 - 1; % Move temp1's index
           p_switch = 2; % After temp1's turn, switch to temp2's turn
        else              % If its temp2's turn in the alternation
           if(isnan(vector2(p2)))
               i = i + 1; % Since NaN is not used, move w's index back
           else
               w(i) = vector2(p2);
           end
           p2 = p2 - 1; % Move temp2's index
           p_switch = 1; % After temp2's turn, switch to temp1's turn
        end
        i = i - 1; % Move w's index
    end
end
```

Results

We have successfully written the algorithm which creates a bivariate polynomial given a sample data set and plots the three-dimensional surface. So far we have only tested a couple functions and have shown the output graph of one of them below.

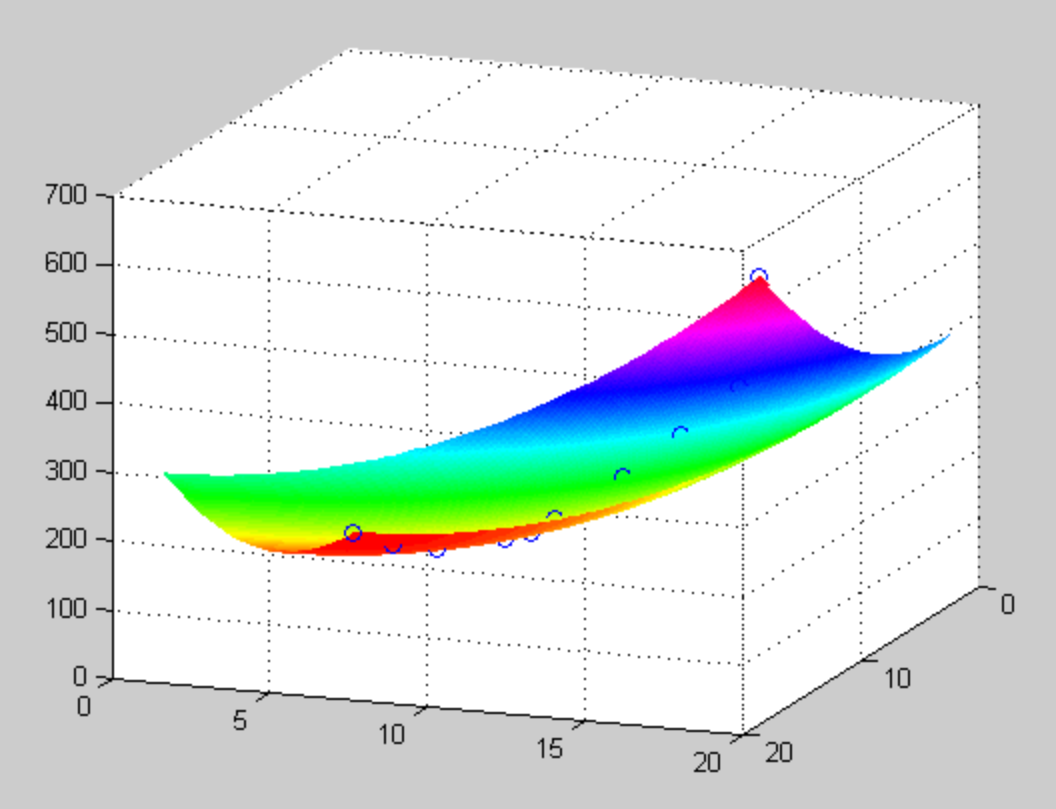


Figure: Graph of the function interpolated from points of the form $f = x^2 + y^2$.

Error analysis

To determine how well a particular function was interpolated, we compare the interpolated values to the actual values of that function. In order to find the error, we take each point in the mesh grid (which represents the X-Y plane) and find the difference of the interpolated *z* value and the actual *z* value at that point in the mesh grid. The absolute value is then taken to give the positive difference of the two values. When analyzing the errors, we will be looking at the average error and the max error.

Both the average error and the max error will be used to judge the interpolations correctness. By this measure, we can consistently analyze where this algorithm works well and where it works poorly. The variables of interest will be: different classes of functions, number of sample points, and distribution of sample points.

[Insert table of errors here]

Conclusion

(to be written)

Error documentation (not part of the report)

11/17/14

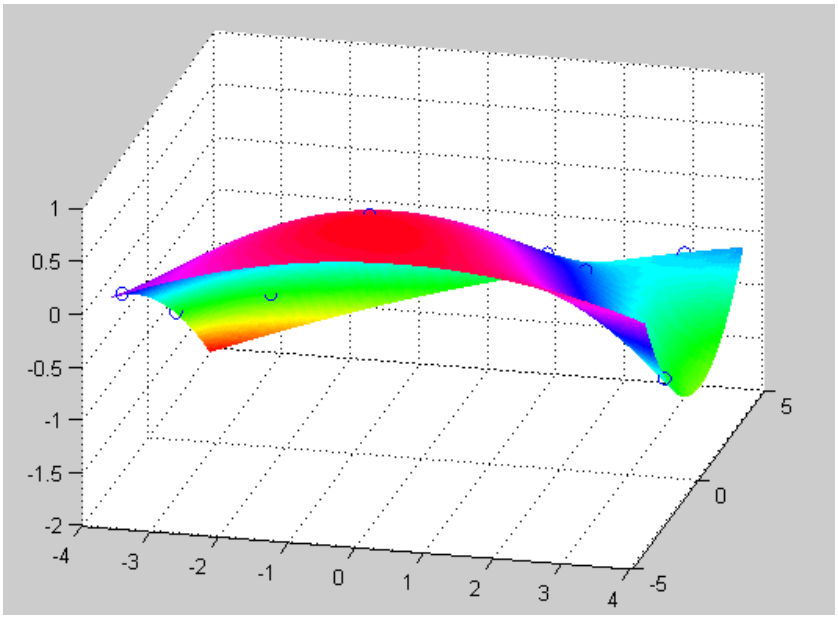
We chose a different function to interpolate using our algorithm today to test how it would perform on different types of functions. The function we chose was

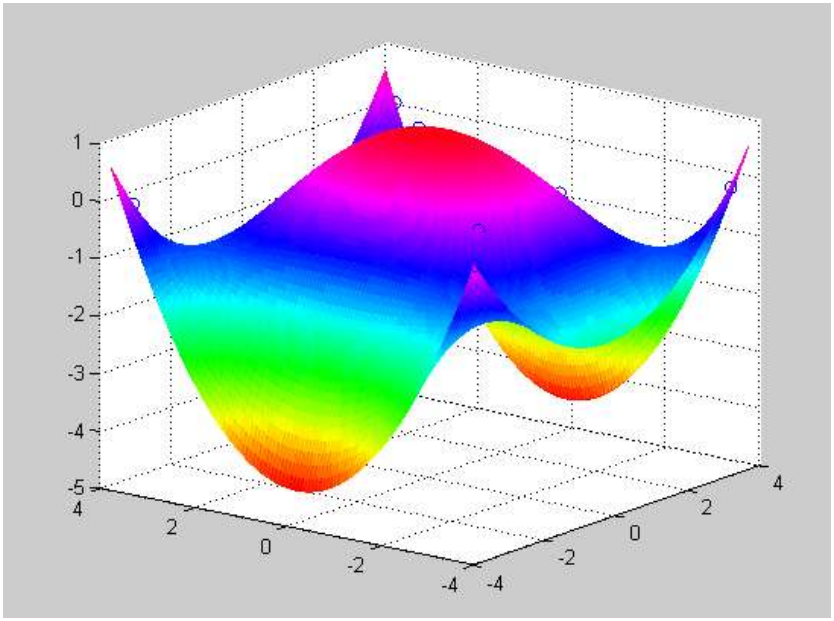
$$z = \sin(1 + \sqrt{x^2 + y^2})$$

We tested different methods of obtaining points using randomly chosen points, a linear point distribution, and strategically picking points. The results from the random points and the linear point distribution was not interpolating the function well which is shown below. Then we realized that we needed to pick the points based on the areas of interest in the graph, like where the changes in curvature are and such. This is where we strategically picked the points based on the different curves in the graph and the result was much better as shown below.

Also, we tried to replicate the exact graph of the function by using a linspace as our data set for x, y, and z. This was a problem because picking a linear combination of points (a linspace create a set of evenly spaced points) creates a non-singular matrix like the problem we were having before. (what if we used a different method of solving the matrix instead of c = z/V? (Gauss).

Also, we were unsure if creating the linspace would have been an accurate interpolation because there would have been a high number of data points which would create a high degree polynomial. However, we were unable to test this because the matrix created with a linspace was singular.


1
0.5
0
-0.5
-1
-1.5
-2
-4
-3
-2
-1
0
1
2
3
4
5
0
-5


1
0
-1
-2
-3
-4
-5
4
2
0
-2
-4
-4
-2
0
2
4