

Multivariate interpolation

Group:

Cameron Moats

The algorithm we will be looking at is multivariate interpolation and we'll specifically look at examples in the 3d case as it's easy to visualize. Multivariate interpolation doesn't work the same as polynomial interpolation, we can't just have some unknown points and get an equation, instead we must start with an equation with some unknown coefficients and some data points, then we can solve for the unknown coefficients to get our multivariate interpolation. So we have to input an equation that we're trying to solve for, one of the advantages of this is the equation can have combinations of things like $\cos(x)$ and e^x in it where we couldn't get that with polynomial interpolation. However, there is a disadvantage of having to make an equation with these unknowns, and it's an important one, the equation will not always be able to solve for the unknown coefficients (because of indeterminates), which means there is no way for the equation entered to work for the data points you entered. With these two things in mind, we can begin our discussion with how the algorithm works. First, the user must determine what data points they want to enter and what equation they want to fit it to. When coming up with an equation you must enter it as a list and each entry in the list must be unique (we can't have $\{x, x\}$) and x, y, z , etc must be entered as $x = y_1, y = y_2, z = y_3$, etc. So, For example, $f(x) = \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 y_1^2 + \alpha_4 y_2^2 + \alpha_5 3$ should be entered as $\{y_1, y_2, y_1^2, y_2^2, 3\}$. I wrote a function that generates an equation if the user does not want to do this themselves, but this equation's coefficients α_i will not always be solvable for certain data points because of indeterminates, so it is not guaranteed to work. For this algorithm to work, the number of data points must equal the number of items in the list because we will create a matrix with dimensions that corresponds to the number of data points and the number of items and then take the determinant of this matrix, but for this determinate to exist we need an $n \times n$ matrix which is why these two lengths are equal to each other. Now that we have some of the more technical aspects of the algorithm explained, we'll do the algorithm but we'll look at it through an example since the symbolic form is pretty strange looking.

$$z_i = \alpha_1 y_{1i} + \alpha_2 y_{2i} + \alpha_3$$

y_1 and y_2

$$z_i = \{y_{1i}, y_{2i}, 1\}$$

$\{y_1, y_2, y_3, \text{etc}\}$

$\{y_1, y_2, y_3, \text{etc}\}$

We want to interpolate the data points $\{(0,0,1),(0,1,2),(1,1,3)\}$ on a function

$z_i = \alpha_1 y_{1_i} + \alpha_2 y_{2_i} + \alpha_3$. So as a list $z_i = \{y_{1_i}, y_{2_i}, 1\}$. Here i just represents the i th data point. Now we'll plug that values y_1 and y_2 from the i th data points into z and keep it as a list. (Note: I did not include the i when talking about how to enter in the equation to the algorithm because the algorithm won't work if you do this, but when you do this on paper you should have $\{y_{1_i}, y_{2_i}, y_{3_i}, \text{etc}\}$ instead of $\{y_1, y_2, y_3, \text{etc}\}$)

For $(0,0,1)$: $y_{1_1} = 0$ and $y_{2_1} = 0$ so $z_1 = \{0,0,1\}$

For $(0,1,2)$: $y_{1_2} = 0$ and $y_{2_2} = 1$ so $z_2 = \{0,1,1\}$

For $(1,1,3)$: $y_{1_3} = 1$ and $y_{2_3} = 3$ so $z_3 = \{1,1,1\}$

Now from these three we can create a matrix out of all the z lists, so our matrix M should be $\{z_1, z_2, z_3\}$

$$M = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{ or } M = \{(0,0,1), (0,1,1), (1,1,1)\}$$

This is what coeffmatrix does in our algorithm and we could do this for more data points as long as the number of data points equals the length of the list of our equation. If we did this for 5 data points we would need to create a 5x5 matrix using the same steps as this case.

Now we will replace each i th row with just z , without having values of the data points plugged into z . We will do this for how many i rows there are, in our case 3, and we will call each of these matrices Δ_i

$$\Delta_1 = \begin{bmatrix} y_1 & y_2 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\Delta_2 = \begin{bmatrix} 0 & 0 & 1 \\ y_1 & y_2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\Delta_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ y_1 & y_2 & 1 \end{bmatrix}$$

Now we need to calculate the determinant of all the matrices we have found and in this example we need to do this 4 times

$$\text{Det}(M) = -1$$

$$\text{Det}(\Delta_1) = y_2 - 1$$

$$\text{Det}(\Delta_2) = y_1 - y_2$$

$$\text{Det}(\Delta_3) = -y_1$$

$$f = \sum_{i=1}^p f_i \frac{\text{Det}(\Delta_i)}{\text{Det}(M)} \quad f_i$$

$$z_1 \left(\frac{\text{Det}(\Delta_1)}{-1} \right) + z_2 \left(\frac{\text{Det}(\Delta_2)}{-1} \right) + z_3 \left(\frac{\text{Det}(\Delta_3)}{-1} \right) = (1 - y_2) + 2(y_2 - y_1) + 3y_1 = y_1 + y_2 + 1$$

$$\Delta_1) = y_2 - 1$$

$$\Delta_2) = y_1 - y_2$$

$$\Delta_3) = -y_1$$

The last thing we do is enter this into a magical equation which is $f = \sum_{i=1}^p f_i \frac{\text{Det}(\Delta_i)}{\text{Det}(M)}$ where f_i should be our

last value in our i th data point and p is our number of data points. so we get $z =$

$$z_1 \left(\frac{\text{Det}(\Delta_1)}{-1} \right) + z_2 \left(\frac{\text{Det}(\Delta_2)}{-1} \right) + z_3 \left(\frac{\text{Det}(\Delta_3)}{-1} \right) = (1 - y_2) + 2(y_2 - y_1) + 3y_1 = y_1 + y_2 + 1$$

We could go about doing the same steps for other equations and data points

```
coeffmatrix[m_, z_] := Module[{dimensions, matrix, npoints, ncoefficients},
  (*m is our data points and z is a list of a function where each object
  in the list corresponds to a single term with unknown coefficients
  For exampe: f=α1y1+α2y2 should be {y1,y2}.
  Note: please enter x,y,z,etc in z as y1,y2,y3,etc or this will not work *)
  dimensions = Length[m[[1]]];
  (*Number of dimenstions the points have. (x,y,z) would=3 *)
  npoints = Length[m];
  ncoefficients = Length[z];
  (*This is just the number of coeffecients we're solving for in z *)
  matrix = {}; (* We want to create a matrix where each
  row takes the value of z when m[[row#]] values are plugged into z *)
  If[ncoefficients ≠ npoints, Print[
    "Number of points must equal the number of coefficeints in your equation"];
  Return["Number of points must equal the number of coefficeints in your equation"],
  (* Theorem states npoints=ncoefficients for this interpolation to work*)
  For[i = 1, i < npoints + 1, i++,
    For[k = 1, k < dimensions + 1, k++, yk = m[[i]][[k]]];
    (* This could calculate one too many yk values since there could be a constant
    term in z, but only one so there's no real reason to work around it *)
    AppendTo[matrix, z]];
  (*This just adds a row in the matrix that corresponds
  to the z values where m[[row#]] are the points plugged into z *)
  For[k = 1, k < dimensions + 1, k++, Unset[yk]];
  (* Unset is like clear but for stuff with subscripts
  Note: I didn't have yk in the module variables
  because in mathematica things with subscripts aren't variables *)
  Return[matrix]]]
```

Out[5]= { {0, 0, 1}, {0, 1, 2}, {1, 1, 3} }

In[6] :=

```
interpolate3d[m_, z_] := Module[{matrix, detMatrix, matrixdim, f, npoints, pointdim},
  matrix = coeffmatrix[m, z];
  (* gets the generalized matrix used in the algorithm*)
  matrixdim = Length[matrix];
  (* Gets m=
```

```

n dim of the matrix (m=n) because that needs to be true for the algorithm to work *)
npoints = Length[m];
pointdim = Length[m[[1]]]; (*dimesnions of a data point *)
detMatrix = Det[matrix]; (* Gets determinant of coeffmatrix*)
f = 0; (*this variable is our interpolating function *)
For[i = 1, i < matrixdim + 1, i++, deltamatrix[i] = ReplacePart[matrix, {i} → z];
  detDelta[i] = Det[deltamatrix[i]]]; (* deltamatrix[i] replaces one
  particular row in coeffmatrix and makes it so that the new row is the list
  from z and then detDelta calculates the determinant of this new matrix.

  We need to do this for each row in the matrix so we loop through it all *)
For[i = 1, i < npoints + 1, i++, lastval[i] = m[[i]][[pointdim]];
  f += Expand[lastval[i] * detDelta[i] / detMatrix]];
(* lastval is just the last number in a point from m,
so for (1,2,3) lastval=3. Then the interpolating formula is
used for each replaced row and that's why we have a for loop.

The interpolation formula for each loop is  $\frac{\text{lastvalue} \cdot \text{Det}(\Delta_i)}{\text{Det}(\text{coeffmatrix})}$  *)
Return[f]

```

```

randpoint[points_] := Module[{m}, m = {};
  For[i = 0, i < points, i++, AppendTo[m,
    {RandomInteger[{-10, 10}], RandomInteger[{-10, 10}], RandomInteger[{-10, 10}]}]];
  (* generates a user defined number of random 3d point*)
  Return[m]]

```

```

genfunclist[terms_] :=
Module[{funcarray}, (* terms is the length of the function list *)
  funcarray = {1}; (* the one is just a constant term. You could view it as  $\alpha_1$  *)
  If[Mod[terms, 2] == 0, (* determines whether we have an even or odd amount of terms*)
    For[i = 1, i < 3, i++, For[k = 1, k < (terms) / 2, k++, AppendTo[funcarray, yik]]];
    (* first for loop gets if it's y1 or y2 and second gets the power of it.





    For an even number of terms, we'll have an even amount of y1 and y2 from this
    loop which means we'll have an odd number of terms when adding constant term*)
    AppendTo[funcarray, y1(terms / 2)];
    (* fixes problem from the above comment by adding a unique y1 to this list. *)
    Return[funcarray],
    For[i = 1, i < 3, i++, For[k = 1, k < 1 + (terms - 1) / 2, k++, AppendTo[funcarray, yik]]];
    (* Does same thing as previous for
    loop just makes it work with an odd number of terms *)
    Return[funcarray]]





```





In[]:=

```
tests = {VerificationTest[coeffmatrix[{{0, 1, 2}, {3, 4, 2}}, {y1, y2, 1}],
  "Number of points must equal the number of coefficeints in your equation",
VerificationTest[coeffmatrix[{{1, 2, 3}, {2, -3, -6}}, {y1, y2}], {{1, 2}, {2, -3}}],
VerificationTest[coeffmatrix[{{0, 0, 1}, {0, 1, 2}, {1, 1, 3}}, {y1, y2, 1}],
  {{0, 0, 1}, {0, 1, 1}, {1, 1, 1}}] ×
  VerificationTest[interpolate3d[{{1, 2, 3}, {2, -3, -6}}, {y1, y2}],  $-\frac{3 y_1}{7} + \frac{12 y_2}{7}$ ],
VerificationTest[interpolate3d[{{0, 0, 1}, {0, 1, 2}, {1, 1, 3}}, {y1, y2, 1}], 1 + y1 + y2],
VerificationTest[genfunclist[5], {1, y1, y1^2, y2, y2^2}]}]
```

Number of points must equal the number of coefficeints in your equation

Out[]:= {TestResultObject[  Outcome: Success
Test ID: None], TestResultObject[  Outcome: Success
Test ID: None]},

TestResultObject[  Outcome: Success
Test ID: None] TestResultObject[  Outcome: Success
Test ID: None],

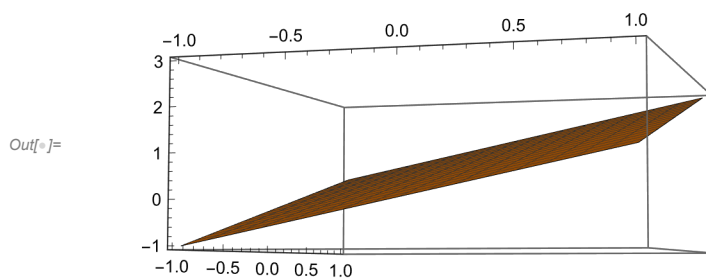
TestResultObject[  Outcome: Success
Test ID: None], TestResultObject[  Outcome: Success
Test ID: None]}

We can now show how the algorithm gets the same result as our example and then graph it (I included it in the test cases)

In[]:= **z = interpolate3d[{{0, 0, 1}, {0, 1, 2}, {1, 1, 3}}, {y1, y2, 1}]**

Out[]:= **1 + y1 + y2**

In[]:= **Plot3D[z, {y1, -1, 1}, {y2, -1, 1}]**



We'll now show when this cannot determine the coefficients of the equation

```
In[ ]:= m = randpoint[8]

{{8, 0, 0}, {8, -9, 7}, {-4, -1, 10}, {6, -3, 4},
{-10, -5, -7}, {-10, -3, -1}, {-10, -8, 1}, {-4, 3, -5}}
m = {{8, 0, 0}, {8, -9, 7}, {-4, -1, 10}, {6, -3, 4},
{-10, -5, -7}, {-10, -3, -1}, {-10, -8, 1}, {-4, 3, -5}}
```

```
In[ ]:= f = genfunclist[8]
```

```
Out[ ]:= {1, y1, y12, y13, y2, y22, y23, y14}
```

```
In[ ]:= interpolate3d[m, f]
```

```
Out[ ]:= Indeterminate
```

It took me about 15 times of rerunning randpoint for this to get an actual indeterminate, which shows that this function works pretty well and rarely does not work. We can also see that the equation cannot be determined when there's an indeterminate.

We can make some really cool looking surfaces when we do this and we'll show one here. We'll use genfunclist to create the function that we're trying to find the unknown coefficients for and we'll use some interestingly picked out data points.

```
In[ ]:= data1 = {{0, 0, -1}, {1, 0, 1}, {0, 1, 1},
{-1, 0, 1}, {0, -1, 1}, {2, 0, 0}, {-2, 0, 0}, {0, 2, 0}, {0, -2, 0}}
```

```
Out[ ]:= {{0, 0, -1}, {1, 0, 1}, {0, 1, 1}, {-1, 0, 1},
{0, -1, 1}, {2, 0, 0}, {-2, 0, 0}, {0, 2, 0}, {0, -2, 0}}
```

```
In[ ]:= {{0, 0, -1}, {1, 0, 1}, {0, 1, 1}, {-1, 0, 1},
{0, -1, 1}, {2, 0, 0}, {-2, 0, 0}, {0, 2, 0}, {0, -2, 0}}
```

```
Out[ ]:= {{0, 0, -1}, {1, 0, 1}, {0, 1, 1}, {-1, 0, 1},
{0, -1, 1}, {2, 0, 0}, {-2, 0, 0}, {0, 2, 0}, {0, -2, 0}}
```

```
In[ ]:= b = genfunclist[9]
```

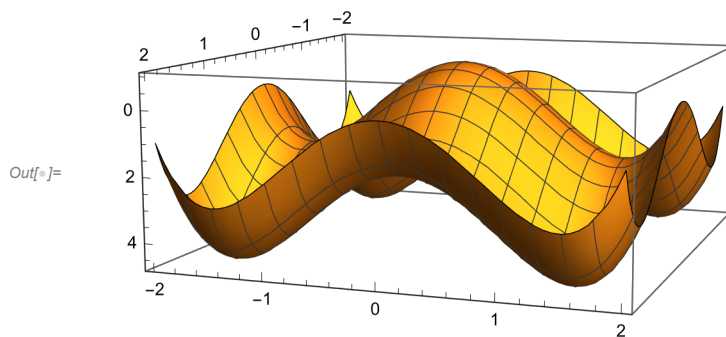
```
Out[ ]:= {1, y1, y12, y13, y14, y2, y22, y23, y24}
```

Now we can find the multivariate interpolation of this

```
In[ ]:= k = interpolate3d[data1, b]
```

```
Out[ ]:= -1 +  $\frac{31 y_1^2}{12}$  -  $\frac{7 y_1^4}{12}$  +  $\frac{31 y_2^2}{12}$  -  $\frac{7 y_2^4}{12}$ 
```

```
In[ ]:= Plot3D[f, {y1, -2, 2}, {y2, -2, 2}]
```



We can check to make sure this actually get's the values we expect based of the data y_1 and y_2 values (we'll let $x=y_1$ and $y=y_2$)

```
In[ ]:= g[x_, y_] = -1 +  $\frac{31 x^2}{12}$  -  $\frac{7 x^4}{12}$  +  $\frac{31 y^2}{12}$  -  $\frac{7 y^4}{12}$ 
```

```
Out[ ]:= -1 +  $\frac{31 x^2}{12}$  -  $\frac{7 x^4}{12}$  +  $\frac{31 y^2}{12}$  -  $\frac{7 y^4}{12}$ 
```

```
In[ ]:= g[0, 0]
```

```
Out[ ]:= -1
```

```
In[ ]:= g[1, 0]
```

```
Out[ ]:= 1
```

```
In[ ]:= g[0, 1]
```

```
Out[ ]:= 1
```

```
In[ ]:= g[-1, 0]
```

```
Out[ ]:= 1
```

```
In[ ]:= g[0, -1]
```

```
Out[ ]:= 1
```

```
In[ ]:= g[2, 0]
```

```
Out[ ]:= 0
```

```
In[ ]:= g[-2, 0]
```

```
Out[ ]:= 0
```

```
In[ ]:= g[0, 2]
```

```
Out[ ]:= 0
```

```
In[ ]:= g[0, -2]
```

```
Out[ ]:= 0
```

So we can see this function works for all data points we listed and it's pretty neat looking too.

I think it's important to note the more data points we try interpolating, it will take significantly longer for the algorithm to run since it takes a long time to calculate determinants of 50 x 50 matrices or higher. I read that the computing time for $n \times n$ matrices is around $O(n^3)$, which means the computation time grows at a rate of n^3 where n is the number dimensions (I think that's what that means. It's big o notation that's all I read). I initially said I was gonna find some applications of this, but all I found was how multivariate interpolation is used to model Earth's surface but couldn't find a paper on it. So overall this interpolation is pretty cool and useful for geostatistics, but it's computationally expensive and must be used cautiously.