

Algebraic Construction of Smooth Interpolants on Polygonal Domains

**Elisabeth Anna Malsch
Gautam Dasgupta**

A smooth and bounded interpolant can be constructed in explicit algebraic form within any polygon, convex or concave. The resulting function is not unique and accordingly can be adjusted to satisfy desired global conditions, such as linear fields. The closed-form representation is obtained by combining simple geometric descriptions, such as the side lengths and areas. The interpolant distributes values given at discrete nodes smoothly over the interior of the domain. On a convex polygon, the interpolant is a rational function of the product of areas. On a concave or multiply connected polygon, the interpolant is a function of areas and edge lengths, which introduces a square root term.

■ Introduction

There are many varied applications for a smooth and bounded interpolant. The most straightforward is for computer graphics, for example in rendering and coloring applications [1]. Smoothly distributing color defined at nodes over a domain requires a bounded interpolating function. Interpolants can also be employed as test functions for computational modeling [2]. For the finite element method, they act as shape functions [3, 4]. Similar interpolants have been applied to the analysis of biomedical growth and shape change [5].

Conventional methods for constructing interpolants do not distribute boundary values smoothly over arbitrary polygonal shapes. Most either require a mesh or apply only to very simple geometries [4, 6]. For example, the conventional finite element development does not specifically address concavity; instead, concave domains are tessellated into convex parts. An exception is the R-function construction that applies to any polygonal domain [7]. Unlike the proposed formulation, it can only represent homogeneous boundary conditions.

In general, any function $u(x, y)$ can be approximated as the sum of the products of the value u_i at given nodes (x_i, y_i) and an interpolant $N_i(x, y)$:

$$u(x, y) \simeq \hat{u}(x, y) = \sum_{i=1}^n u_i N_i(x, y). \quad (1)$$

The derived interpolants $N_i(x, y)$ are linearly independent:

$$N_i(x_j, y_j) = \delta_{i,j}. \quad (2)$$

Along any given boundary, only the values given at the end nodes prescribe the behavior along the line segment:

$$\hat{u}(t x_i + (1-t) x_j, t y_i + (1-t) y_j) = f(u_i, u_j, t). \quad (3)$$

Within the domain, the interpolants are bounded, normalized between zero and one, and smooth. The interpolant generated is not unique. Any function that satisfies an elliptic operator and the boundary conditions satisfies the interpolant requirements.

Using the same method to construct general bounded interpolants on any polygon, linear interpolants are constructed. A linear interpolant satisfies the additional requirements of boundary linearity:

$$\hat{u}(t x_i + (1-t) x_j, t y_i + (1-t) y_j) = t u_i + (1-t) u_j, \quad (4)$$

and exact reproduction of linear fields:

$$x = \sum_{i=1}^n x_i N_i(x, y) \quad \text{and} \quad y = \sum_{i=1}^n y_i N_i(x, y). \quad (5)$$

The constant field is satisfied by construction:

$$1 = \sum_{i=1}^n N_i(x, y). \quad (6)$$

In most cases, the linearity requirements do not prescribe a unique interpolant. Therefore additional global conditions, such as higher-order fields, could also be imposed.

■ Input Data

The interpolant can be constructed on any two-dimensional slice of any polytope, including polygons, polyhedra, or higher-dimensional objects. Accordingly, the dimension of the data must be defined. The examples are in two dimensions.

```
In[1]:= xy = {x, y};
```

The list of vertices must also be defined; for example, here is a triangle with three nodes.

```
In[2]:= vertexConnect = {{1, 2, 3}};
```

Another example is a triangle enclosing a quadrilateral and an interior point.

```
In[3]:= vertexConnect = {{1, 2, 3}, {4}, {8, 7, 6, 5}};
```

For a simple interpolant, the quantity that is going to be distributed smoothly over the domain must be defined at every nodal point. The nodal points not only determine the structure of the domain, they are also the points at which the value being distributed is given. The vertices of the polygon are a subset of the nodal points. Nodal points and interpolation points are one and the same.

■ Geometric Measures and Support Functions

It is useful to construct the interpolant with respect to geometric measures as opposed to nodal locations. The benefit of such an approach is that it is invariant of coordinate system and dimension.

□ The Distance Function

The distance between two points p_1 and p_2 with coordinates (x_1, y_1) and (x_2, y_2) is:

$$|p_1 p_2| = l_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \quad (7)$$

The function can be written as the norm of the vector connecting the points, $\text{norm}[p_1 - p_2]$. The built-in `Norm` function would be appropriate if the points were represented as complex numbers rather than as pairs of real numbers.

```
In[4]:= norm[a_] := Sqrt[a.a]
```

□ The Signed Area of a Triangle

The area of the triangle with vertices p_1, p_2, p_3 with coordinates $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ can be defined in terms of the determinant:

$$A = \Delta[p_1, p_2, p_3] = 1/2 \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}. \quad (8)$$

The area is signed: $\Delta[p_1, p_2, p_3] = -\Delta[p_2, p_1, p_3]$.

```
In[5]:= area[p_, q_, r_] :=
  1 / 2 Det[Join[Transpose[{p, q, r}], {{1, 1, 1}}]] /;
  Length[p] == Length[q] == Length[r] == 2
```

□ The Unsigned Area Function

The unsigned area of a triangle can be written in terms of side lengths using Heron's formula:

$$A = \sqrt{s(s-a)(s-b)(s-c)}, \quad (9)$$

where

$$s = \frac{1}{2}(a+b+c). \quad (10)$$

The resulting area is necessarily nonnegative. The area measure has no physical meaning for a node located on the edge of the domain. In such a case, defining the unsigned area to be one when the signed area is zero allows the concave element shape function routines to apply to elements with side nodes.

```

In[6]:= unsignedArea[p_, q_, r_] := Module[{a, b, c, s},
  a = norm[p - q]; b = norm[q - r]; c = norm[r - p]; s = 1/2 (a + b + c);
  Sqrt[s (s - a) (s - b) (s - c)] ]
In[7]:= unsignedArea[p_, q_, r_] := 1 / ; area[p, q, r] == 0

```

□ The Angle Functions

The cosine or sine of an angle defined by three vertices can be determined using line lengths and the unsigned area. Care is taken to choose the proper sign for the cosine and sine functions. For the angle β defined by the nodes p, q, r with the angle at q :

$$\cos(\beta) = \frac{|pq|^2 + |qr|^2 - |rp|^2}{2|pq||qr|}, \quad (11)$$

$$\sin(\beta) = \frac{2|\Delta[p, q, r]|}{|pq||qr|}. \quad (12)$$

The angle β never has to be evaluated.

```

In[8]:= cosAng[p_, q_, r_] := Module[{pq, qr, rp}, pq = norm[p - q];
  qr = norm[q - r]; rp = norm[r - p]; (pq^2 + qr^2 - rp^2) / (2 pq qr) ]
In[9]:= sinAng[p_, q_, r_] := 2 unsignedArea[p, q, r] / (norm[p - q] norm[q - r])

```

■ Minimum Functions

The geometric definitions can be used to define minimum functions. For example, the distance from node p to node q , $|pq|$, is zero and minimized when $p = q$ and the area, defined by the three nodes p, q, r , is zero and minimized when p is on the line qr . The R-function formulation is similarly dependent on minimum functions [7].

Define a function of the variable point $z(x, y) = \{x, y\}$ that is strictly zero along the line qr :

$$|z(x, y) - q| + |z(x, y) - r| - |q - r|. \quad (13)$$

The equation applies in any dimension.

```
In[10]:= perim[xy_, q_, r_] := norm[xy - q] + norm[xy - r] - norm[q - r]
In[11]:= Show[#, DisplayFunction -> $DisplayFunction,
  ImageSize -> {200, Automatic}, PlotRange -> All] & /@
  Thread[{{ContourPlot[#[[x, y], {-1/2, -1/2}, {1/2, 1/2}],
    {x, -1, 1}, {y, -1, 1}, DisplayFunction -> Identity,
    PlotPoints -> 33, PlotRange -> {0, 1}] & /@
    {norm[#1 - {0, 0}] &, area, unsignedArea, perim},
  {Graphics[{PointSize[.02], RGBColor[0, 0, 1], Point[{0, 0}]}],
  Graphics[{PointSize[.02], RGBColor[0, 0, 1],
    Line[{{-1, -1}, {1, 1}}]}]}, Graphics[
  {PointSize[.02], RGBColor[0, 0, 1], Line[{{-1, -1}, {1, 1}}]}]},
  Graphics[{PointSize[.02], RGBColor[0, 0, 1],
    Line[{{-1/2, -1/2}, {1/2, 1/2}}]}]}]}]
```

From In[11]:=

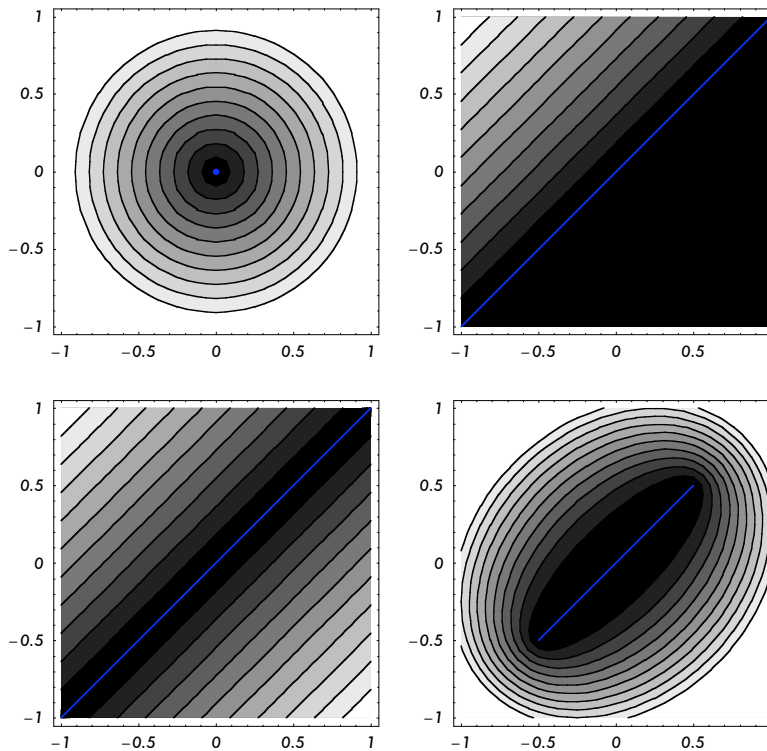


Figure 1. Contours that are minimum at a point, along a line, and along a line segment.

□ Linearity

The perimeter function can be modified so that the contour lines are linear along a broken line segment. Four nodal points are required: say, o , p , q , r . Construct a function that is zero along the path pq and linear along the paths op and qr , with angles $\alpha = \angle opq$ and $\beta = \angle pqr$. Again define the variable point $z(x, y) = \{x, y\}$:

$$A(x, y) = |pq|^2 - 2 \cos(\beta) |qz(x, y)| |pq| - |pz(x, y)|^2 + |qz(x, y)|^2, \quad (14)$$

$$B(x, y) = |pq|^2 - 2 \cos(\alpha) |pz(x, y)| |pq| + |pz(x, y)|^2 - |qz(x, y)|^2. \quad (15)$$

The functions $c_1(x, y)$, $c_2(x, y)$, and $c_3(x, y)$ are defined in terms of $A(x, y)$ and $B(x, y)$:

$$c_1(x, y) = \frac{1}{2} (A(x, y) \sin(\alpha) + B(x, y) \sin(\beta)) |pq|, \quad (16)$$

$$c_2(x, y) = A(x, y) + B(x, y), \quad (17)$$

$$c_3(x, y) = (A(x, y) \cos(\alpha) + B(x, y) \cos(\beta)) |pq|. \quad (18)$$

Here we combine all the pieces:

$$\frac{c_1(x, y) (|pz(x, y)| + |z(x, y)q|)^2 - |pq|^2}{c_2(x, y) (|pz(x, y)| + |z(x, y)q|) - c_3(x, y)}. \quad (19)$$

```

In[12]:= perimLin[xy_, o_, p_, q_, r_] :=
Module[{A, B, pq, xyq, pxy, c1, c2, c3},
  pq = norm[p - q];
  xyq = norm[xy - q];
  pxy = norm[p - xy];
  A = pq^2 - 2 cosAng[p, q, r] xyq pq - pxy^2 + xyq^2;
  B = pq^2 - 2 cosAng[o, p, q] pxy pq + pxy^2 - xyq^2;
  c1 = 1/2 (A sinAng[o, p, q] + B sinAng[p, q, r]) pq;
  c2 = A + B;
  c3 = (A cosAng[o, p, q] + B cosAng[p, q, r]) pq;
  c1 ((xyq + pxy)^2 - pq^2) / ((xyq + pxy) c2 - c3)

```

```

In[13]:= Show[{ContourPlot[Evaluate[
    perimLin[{x, y}, #[[1]], {-1/2, 0}, {1/2, 0}, #[[2]]],
    {x, -1, 1}, {y, -1, 1}, DisplayFunction -> Identity,
    PlotPoints -> 40], Graphics[{RGBColor[1, 0, 0],
    Line[#[[1]], {-1/2, 0}], Line[{1/2, 0}, #[[2]]],
    RGBColor[0, 0, 1], Line[{-1/2, 0}, {1/2, 0}]}]],
    DisplayFunction -> $DisplayFunction, ImageSize -> {200, Automatic},
    PlotRange -> {{-1, 1}, {-1, 1}}] & /@
    {{{-1, -2}, {1, 1/2}}, {0, 1}, {1/10, -1/10}}}]

```

From In[13]:=

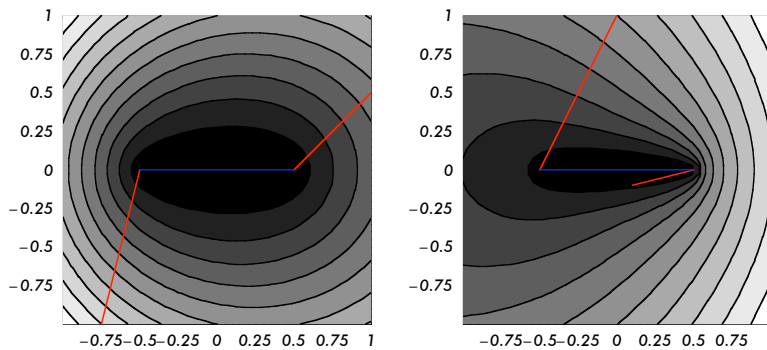


Figure 2. Contours of a function that are minimized along a line segment (in blue) and linear along two adjacent broken line segments (in red).

■ Functions for Displaying Interpolants

```

In[14]:= outline[pts_] := Line[Append[#, #[[1]]] & [pts]]
name[pts_] :=
  Table[Text[ToString[i], pts[[i]], {-2, 0}, Background -> Hue[.2],
    TextStyle -> {FontWeight -> "Bold"}], {i, Length[pts]}]
nodes[pts_] := Point /@ pts
mask[pts_] :=
  Polygon[Flatten[Append[#, #[[1]]] & /@ {{{-1, -1}, {-1, 1},
    {1, 1}, {1, -1}} + ({#[[1]], #[[2]], #[[4]], #[[3]]} & [
    Flatten[Outer[List, ##] & @ @
      ({Min[#, Max[#]} & /@ Thread[pts]), 1]]), pts], 1]]
output[i_, colorfunction_, f_, points_, cnct_, j_: 1] :=
  Module[{fc, pContour, pts}, {
    pts = Part[points, cnct[[1]]];
    fc = Compile[{{x, _Real}, {y, _Real}},
      Evaluate[f[{x, y}, points, cnct, j]]];
    pContour = ContourPlot[Check[fc[x, y][[i]],
      0., CompiledFunction::"cfn"],
      {x, Min[Thread[points][[1]]], Max[Thread[points][[1]]]},
      {y, Min[Thread[points][[2]]], Max[Thread[points][[2]]]},

```

```

DisplayFunction → Identity, (*ContourShading→False*)
ColorFunction → colorfunction, Contours → 20,
PlotPoints → 60, PlotRange → {0, 1}]; Show[pContour,
Graphics[Flatten[{{RGBColor[1, 1, 1]}, {mask[pts]}},
Polygon /@ (points[[#]] & /@ Drop[cnct, 1])]]],
Graphics[Flatten[{{PointSize[.02]}, {outline /@
(points[[#]] & /@ cnct), name[points], nodes[points]}]}],
PlotRange → All, Frame → False, AspectRatio → 1]]]

```

■ Interpolants for Convex Polygons

The interpolant is constructed as a combination of the geometric measures.

```

In[19]:= convex[{x_, y_}, points_, cnct_, i_] :=
Module[{pts = Part[points, cnct[[i]]],
areaTog, n, varArea, zeroArea}, n = Length[pts];
areaTog = Apply[area, Partition[pts, 3, 1, {2}], {1}];
varArea =
Apply[area[##, {x, y}] &, Partition[pts, 2, 1, {1}], {1}];
zeroArea = Apply[Times, Partition[varArea,
Length[pts] - 2, 1, {Length[pts]}], {1}];
(zeroArea * areaTog) / (zeroArea.areaTog)

```

Interpolants are rational functions. For example, here is the set of functions for a triangle, rectangle, skew quadrilateral, and pentagon.

```

In[20]:= convex[{x, y}, {{0, 0}, {1, 0}, {1, 1}}, {{1, 2, 3}}, 1] // Simplify

```

```

Out[20]= {1 - x, x - y, y}

```

```

In[21]:= convex[{x, y}, {{0, 0}, {1, 0}, {1, 1/2}, {0, 1/2}},
{{1, 2, 3, 4}}, 1] // Simplify

```

```

Out[21]= {(-1 + x) (-1 + 2 y), x - 2 x y, 2 x y, -2 (-1 + x) y}

```

```

In[22]:= convex[{x, y}, {{0, 0}, {1, 0}, {3/2, 1}, {-1/2, 1/4}},
{{1, 2, 3, 4}}, 1] // Simplify

```

```

Out[22]= { - (7 + 6 x - 16 y) (-2 + 2 x - y) / (14 + 12 x + 59 y),
2 (7 + 6 x - 16 y) (x + 2 y) / (14 + 12 x + 59 y), 26 y (x + 2 y) / (14 + 12 x + 59 y), - 28 (-2 + 2 x - y) y / (14 + 12 x + 59 y) }

```

```

In[23]:= convex[{x, y}, {{0, 0}, {1, 0}, {3/2, 1/2},
{-1/4, 1}, {-1/2, 1/4}}, {Range[5]}, 1] // Simplify

```

$$\text{Out}[23]= \left\{ \begin{aligned} & \frac{(7 + 12x - 4y)(-1 + x - y)(-13 + 4x + 14y)}{91 - 48x^2 + 487y + 198y^2 + 2x(64 + 33y)}, \\ & - \frac{(7 + 12x - 4y)(x + 2y)(-13 + 4x + 14y)}{91 - 48x^2 + 487y + 198y^2 + 2x(64 + 33y)}, \\ & \frac{18(7 + 12x - 4y)y(x + 2y)}{91 - 48x^2 + 487y + 198y^2 + 2x(64 + 33y)}, \\ & - \frac{184(-1 + x - y)y(x + 2y)}{91 - 48x^2 + 487y + 198y^2 + 2x(64 + 33y)}, \\ & \frac{28(-1 + x - y)y(-13 + 4x + 14y)}{91 - 48x^2 + 487y + 198y^2 + 2x(64 + 33y)} \end{aligned} \right\}$$

```
In[24]:= Show[GraphicsArray[
  Partition[Graphics[#][[1, 1]], PlotRange -> All, AspectRatio -> 1] & /@
  {output[1, (RGBColor[1, (1 - #), (1 - #)] &),
    convex, {{0, 0}, {1, 0}, {1, 1}}, {{1, 2, 3}}},
  output[1, (RGBColor[(1 - #), 1, (1 - #)] &), convex,
    {{0, 0}, {1, 0}, {1, 1/2}, {0, 1/2}}, {{1, 2, 3, 4}}},
  output[1, (RGBColor[(1 - #), (1 - #), 1] &), convex,
    {{0, 0}, {1, 0}, {3/2, 1}, {-1/2, 1/4}}, {{1, 2, 3, 4}}},
  output[1, (RGBColor[1, (1 - #), 1] &), convex, {{0, 0}, {1, 0},
    {3/2, 1/2}, {-1/4, 1}, {-1/2, 1/4}}, {Range[5]}]},
  2]], ImageSize -> {350, Automatic}]
```

From In[24]:=

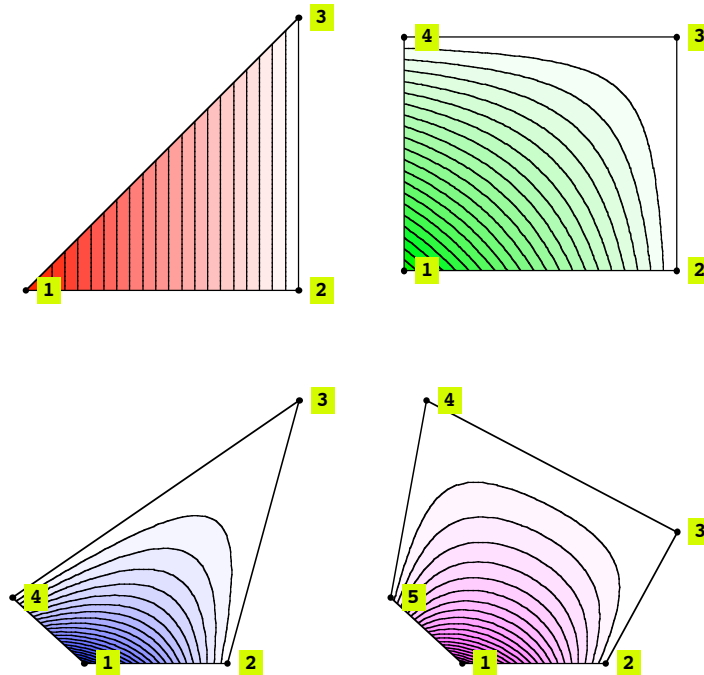


Figure 3. Interpolants on a triangle, rectangle, skew quadrilateral, and pentagon.

■ Interior Points

Points inside the domain are dealt with using the distance function.

```
In[25]:= interior[{x_, y_}, points_, cnct_, 1] :=
Module[{pts = points[[Flatten[Select[cnct, (Length[#] == 1 &)]]]],
  zeroDist = Apply[Times, Partition[(norm[{x, y} - #] & /@ pts),
    Length[pts] - 1, 1, {Length[pts]}], {1}];
  zeroDist / (Plus@@zeroDist)]
```

```
In[26]:= points = {{-.1, -.1}, {1.1, -.1},
  {1.1, 1.1}, {-.1, 1.1}, {0, 0}, {1, 1}, {1, 0}};
cnct = {{1, 2, 3, 4}, {5}, {6}, {7}};
interior[{x, y}, points, cnct, 1][[1]]
Show[output[1, (RGBColor[1, 1, (1 - #)] &), interior, points, cnct],
  DisplayFunction -> $DisplayFunction]
```

$$\text{Out[28]} = \frac{\left(\sqrt{(-1+x)^2 + (-1+y)^2} \sqrt{(-1+x)^2 + y^2}\right)}{\left(\sqrt{(-1+x)^2 + (-1+y)^2} \sqrt{(-1+x)^2 + y^2} + \sqrt{(-1+x)^2 + (-1+y)^2} \sqrt{x^2 + y^2} + \sqrt{(-1+x)^2 + y^2} \sqrt{x^2 + y^2}\right)}$$

From In[26]:=

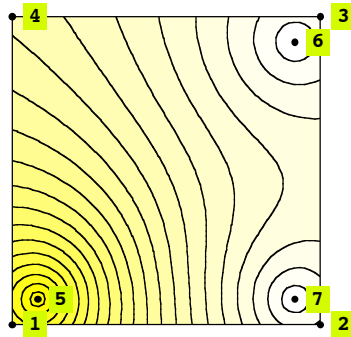


Figure 4. Interior points.

■ Concave Polygons

For concave polygons, the `zeroArea[a,b,c]` function is replaced with the `zeroPerim[a,b,c]` function.

```
In[30]:= varPerim[xy_, points_, cnct_, i_] :=
Module[{pts = Part[points, cnct[[i]]]},
  Apply[perim[xy, ##] &, Partition[pts, 2, 1, {1}], {1}];
```

```
In[31]:= concave[{x_, y_}, points_, cnct_, i_] :=
Module[{pts = Part[points, cnct[[i]]], n, zeroPerim},
n = Length[pts];
zeroPerim = Apply[Times,
Partition[varPerim[{x, y}, pts, cnct, i], n - 2, 1, {n}], {1}];
(zeroPerim) / Apply[Plus, zeroPerim]
```

The shape functions for a concave quadrilateral can easily be constructed. The interpolant contains a square root. The branch points of the square root terms add a discontinuity in the gradient of the interpolant. This discontinuity is necessary for the modeling of re-entrant corners.

```
In[32]:= points = {{-1, -1}, {0, 0}, {1, -1}, {0, 1}};
cnct = {{1, 2, 3, 4}};
concave[{x, y}, points, cnct, 1] // Simplify
```

```
Out[34]= { ( ( (-sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt((-1 + x)^2 + (1 + y)^2)
(-sqrt(2) + sqrt(x^2 + y^2) + sqrt((-1 + x)^2 + (1 + y)^2) ) ) /
(-sqrt(2) - sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt(x^2 + y^2) +
sqrt((-1 + x)^2 + (1 + y)^2) + sqrt((1 + x)^2 + (1 + y)^2) )^2,
( (-sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt((-1 + x)^2 + (1 + y)^2)
(-sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt((1 + x)^2 + (1 + y)^2) ) ) /
(-sqrt(2) - sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt(x^2 + y^2) +
sqrt((-1 + x)^2 + (1 + y)^2) + sqrt((1 + x)^2 + (1 + y)^2) )^2,
( (-sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt((1 + x)^2 + (1 + y)^2)
(-sqrt(2) + sqrt(x^2 + y^2) + sqrt((1 + x)^2 + (1 + y)^2) ) ) /
(-sqrt(2) - sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt(x^2 + y^2) +
sqrt((-1 + x)^2 + (1 + y)^2) + sqrt((1 + x)^2 + (1 + y)^2) )^2,
( (-sqrt(2) + sqrt(x^2 + y^2) + sqrt((-1 + x)^2 + (1 + y)^2)
(-sqrt(2) + sqrt(x^2 + y^2) + sqrt((1 + x)^2 + (1 + y)^2) ) ) /
(-sqrt(2) - sqrt(5) + sqrt(x^2 + (-1 + y)^2) + sqrt(x^2 + y^2) +
sqrt((-1 + x)^2 + (1 + y)^2) + sqrt((1 + x)^2 + (1 + y)^2) )^2 } }
```

```
In[35]:= Show[GraphicsArray[
  Graphics[#[[1, 1]], PlotRange -> All, AspectRatio -> 1] & /@
  {output[1, (RGBColor[1, (1 - #), (1 - #)] &), concave,
    {{-1, -1}, {0, 0}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}],
  output[1, (RGBColor[1, 1, (1 - #)] &), concave,
    {{0, 1/2}, {1, 0}, {3/2, 1/2}, {-1/4, 1}, {-1/2, 1/4}},
    {{1, 2, 3, 4, 5}}]}], ImageSize -> {350, Automatic}]
```

From In[35]:=

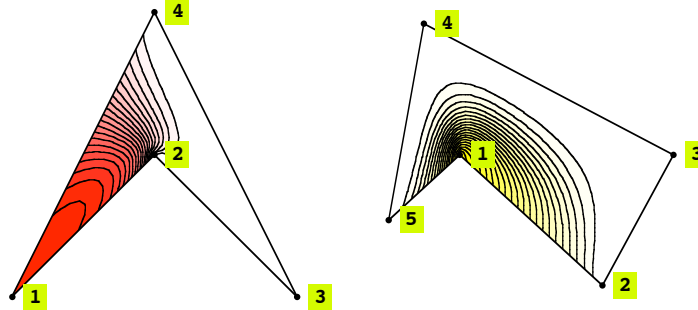


Figure 5. Interpolants on concave polygons do not require tessellation.

■ Concave Polygons with Linearity on the Sides

The $\text{perim}[a, b, c]$ function is replaced with $\text{perimLin}[d, a, b, c, e]$. Similarly the $\text{varPerim}[x, y]$ function is modified so that four vertices define the behavior of the minimum along a line. The constructed interpolants on the quadrilateral and pentagon are linear on the sides.

```
In[36]:= concaveLin[{x_, y_}, points_, cnct_, i_] :=
  Module[{pts = Part[points, cnct[[i]]], zeroPLin, varPLin, areaTog},
    areaTog = Apply[unsignedArea, Partition[pts, 3, 1, {2}], {1}];
    n = Length[pts];
    varPLin =
      Apply[perimLin[{x, y}, ##] &, Partition[pts, 4, 1, {2}], {1}];
    zeroPLin = Apply[Times, Partition[varPLin,
      Length[pts] - 2, 1, {Length[pts]}], {1}];
    (zeroPLin * areaTog) / (areaTog * zeroPLin)]
```

The interpolants for a quadrilateral can be constructed as follows.

```
In[37]:= Show[GraphicsArray[
  Graphics[#[[1, 1]], PlotRange -> All, AspectRatio -> 1] & /@
  {output[1, (RGBColor[(1 - #), (1 - #), 1] &), concaveLin,
    {{-1, -1}, {0, 0}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}},
  output[1, (RGBColor[(1 - #), 1, (1 - #)] &), concaveLin,
    {{0, 1/2}, {1, 0}, {3/2, 1/2}, {-1/4, 1}, {-1/2, 1/4}},
    {{1, 2, 3, 4, 5}}]}], ImageSize -> {350, Automatic}]
```

From In[37]:=

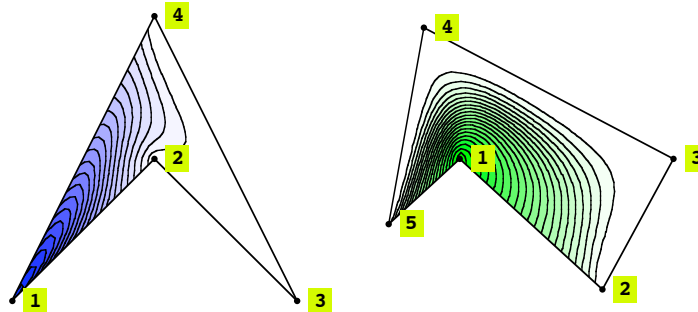


Figure 6. Interpolants that are smooth and linear on the sides of a concave polygon.

The formulation for a concave element applies to any shape. For linearity on a side the area function must be modified.

```
In[38]:= unsignedArea[a_, b_, c_] := 1 /; Abs[area[a, b, c]] < 10^-6
```

Otherwise, the side node would cause singular behavior along the boundary.

```
In[39]:= points = {{-1, -1}, {-1/3, -1},
  {0, -1}, {1/5, -1}, {1, -1}, {1/2, 0}, {0, 1}};
cnct = {Range[Length[points]]};
Plot[Evaluate[
  concaveLin[t points[[1]] + (1 - t) points[[4]], points, cnct, 1]],
  {t, 0, 1}, PlotRange -> All]
```

From In[39]:=

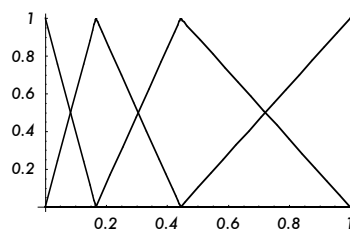


Figure 7. Linearity on the sides.

This works even with multiple side nodes.

```
In[42]:= Show[GraphicsArray[
Graphics[#[[1, 1]], PlotRange -> All, AspectRatio -> 1] & /@
{output[3, (RGBColor[(1 - #), 1, 1] &), concaveLin,
  {{-1, -1}, {0, -1}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}},
output[5, (RGBColor[1, (1 - #), 1] &), concaveLin,
  {{-1, -1}, {-1/3, -1}, {0, -1}, {1/5, -1}, {1, -1}, {1/2, 0},
  {0, 1}}, {Range[7]}]}], ImageSize -> {350, Automatic}]
```

From In[42]:=

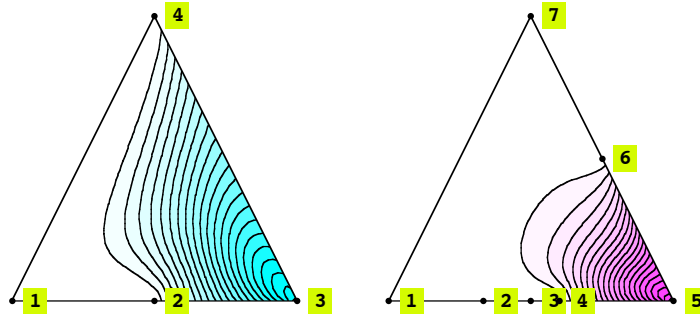


Figure 8. Side node interpolants that are smooth and linear on the sides.

■ Combining Parts

The boundary functions and interior point behaviors can be combined to describe shapes with holes or interior points. Shape functions that apply to separate boundaries and points can be combined.

```
In[43]:= combPrt[{x_, y_}, points_, cnct_, convxH_] :=
Module[{funs, zero, xy = {x, y}}, funs = Table[Switch[cnct[[i]],
a_? (# == convxH &), convex[xy, points, cnct, i],
a_? (Length[#] != 1 &), concaveLin[xy, points, cnct, i],
_, {1}], {i, Length[cnct]}];
zero = Apply[Times, Partition[Table[Apply[Times,
(If[Length[cnct[[i]]] != 1, varPerim[xy, points, cnct, i],
norm[xy - points[[cnct[[i]]][[1]]]]), {i,
Length[cnct]}], Length[cnct] - 1, 1, {Length[cnct]}], {1}];
Flatten[(funs * zero) / Apply[Plus, Flatten[funs zero]]]
```

For example, construct a set of interpolants on a multiply connected domain that contains a triangle and two interior points.

```
In[44]:= Show[GraphicsArray[Partition[
Graphics[#[[1, 1]], PlotRange -> All, AspectRatio -> 1] & /@ {
output[1, (RGBColor[1, 2/3 (1 - #), (1 - #)] &), combPrt,
{{-1, -1}, {1, -1}, {1, 1}, {-1, 1}, {-1/2, -1/2}, {1/2,
-1/2}, {0, 1/2}}, {{1, 2, 3, 4}, {5, 6, 7}}, {1, 2, 3, 4}],
output[8, (RGBColor[7/8 (1 - #), 1, (1 - #)] &), combPrt,
{{-1, -1}, {1, -1}, {1, 1}, {-1, 1}, {-1/2, -1/2},
{1/2, -1/2}, {0, 1/2}, {1/2, 1/2}, {-1/2, 1/2}},
{{1, 2, 3, 4}, {5, 6, 7}, {8}, {9}}, {1, 2, 3, 4}],
output[8, (RGBColor[(1 - #), 1, (2/3) (1 - #)] &),
combPrt, {{-1, -1}, {0, -4/5}, {1, -1}, {1, 1}, {-1, 1},
{-1/2, -1/2}, {0, 0}, {1/2, -1/2}, {0, 1/2}},
{{1, 2, 3, 4, 5}, {6, 7, 8, 9}}, {1, 2, 3, 4}],
output[7, (RGBColor[7/8 (1 - #), 5/6 (1 - #), 1] &), combPrt,
{{-1, -1}, {1, -1}, {1, 1}, {-1, 1}, {-3/4, -3/4},
{-1/4, -3/4}, {-1/4, -1/4}, {-3/4, -1/4},
{3/4, 3/4}, {1/4, 3/4}, {1/4, 1/4}, {3/4, 1/4}},
{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}, {1, 2, 3, 4}],
2]], ImageSize -> {350, Automatic}]
```

From In[44]:=

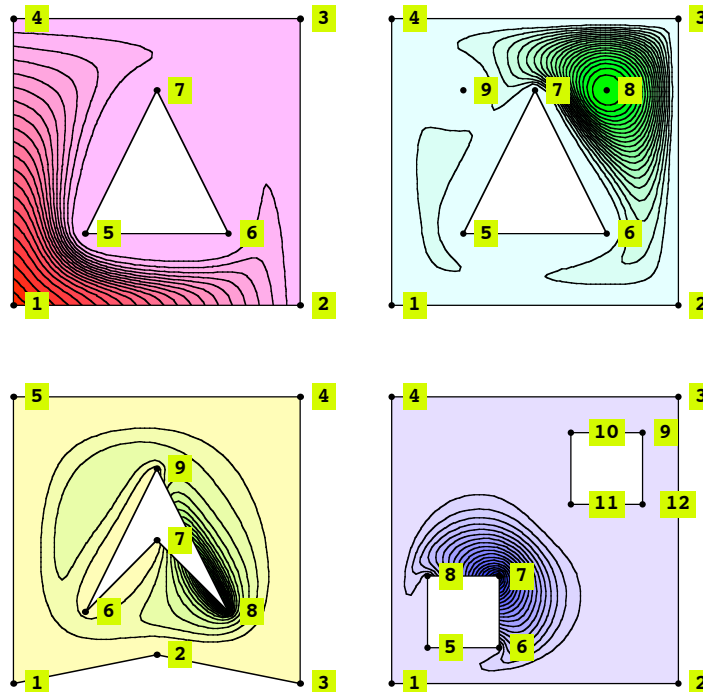


Figure 9. Interpolants that are smooth and linear on the sides of multiply connected polygons.

■ Enforcing Global Linearity

If some of the interpolating functions do not satisfy the same linearity conditions that the convex shape functions satisfy, they can be altered. Given the entire set of interpolating functions $N_j(x, y)$ corresponding to the n nodal points (x_j, y_j) and the set of functions $R_i(x, y)$ that correspond to the points (x_i, y_i) on the convex hull, the following requirement can be imposed:

$$\sum_{j=1}^n a(x_j, y_j) N_j(x, y) = a(x, y). \quad (20)$$

When the function $a(x, y)$ is either 1, x , or y , the convex interpolant satisfies the requirement exactly over the m convex hull points:

$$\sum_{i=1}^m a(x_i, y_i) R_i(x, y) = a(x, y). \quad (21)$$

The functions $N_i(x, y)$ can be modified to ensure linearity:

$$N_i(x, y) = R_i(x, y) - \sum_{k=1}^p R_i(x_k, y_k) N_k(x, y). \quad (22)$$

When the points (x_j, y_j) are not the convex hull points and $p = n - m$, boundedness is not necessarily preserved.

```
In[45]:= makeLin[{x_, y_}, points_, cnct_, cnvxH_] := Module[{nonLin, lin},
  {nonLin = combPrt[{x, y}, points, cnct, cnvxH];
  lin = convex[{x, y}, points, {cnvxH}, 1] -
    (ReplacePart[combPrt[{x, y}, points, cnct, cnvxH], 0, Map[List,
      cnvxH]) . (Map[convex[#, points, {cnvxH}, 1] &, points])};
  Fold[ReplacePart[#1, lin[#2]], #2] &, nonLin,
  Range[Length[cnvxH]]][[1]]]
```

For example, construct linearized interpolations on a quadrilateral.

```

In[46]:= Show[GraphicsArray[
  Graphics[#[[1, 1]], PlotRange -> All, AspectRatio -> 1] & /@
  {output[1, (RGBColor[1, (1 - #), (1 - #)] &), makeLin,
    {{-1, -1}, {0, 0}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}},
  output[1, (RGBColor[(1 - #), (1 - #), 1] &), makeLin,
    {{0, 1/2}, {1, 0}, {3/2, 1/2}, {-1/4, 1}, {-1/2, 1/4}},
    {Range[5]}]}], ImageSize -> {350, Automatic}]

```

From In[46]:=

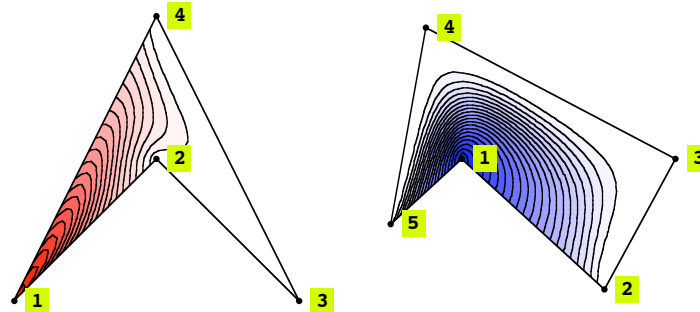


Figure 10. Linearized interpolants.

■ Conclusion

The resulting interpolants are smooth and bounded within their respective domains and the sum of any set of interpolants is one. The functions are linearly independent. Examples of the smooth and bounded behavior of the functions are shown in the figures. The convex polygon shape functions automatically satisfy constant and linear fields. This is not the case for the concave or multiply connected domain representation. Nevertheless, the linearity constraint can be imposed on the representation.

This type of interpolant is distinctly different from the available interpolant and shape function formulations, since each interpolant is constructed to satisfy the required conditions exactly: smoothness, boundedness, linearity on sides, and the linear field conditions. Using this algebraic construction, closed-form interpolants satisfying different field properties can also be derived.

■ Acknowledgment

This research was supported by a National Science Foundation grant (CMS:0202232). We are grateful to the reviewers for their careful reading and many helpful comments.

■ References

- [1] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics Principles and Practice*, 2nd ed., New York: Addison-Wesley, 1996.
- [2] E. L. Wachspress, *A Rational Finite Element Basis*, New York: Academic Press, 1975.
- [3] R. Courant, "Variational Methods for the Solution of Problems of Equilibrium and Vibrations," *Bulletin of the American Mathematical Society*, **49**, 1943 pp. 1–43.
- [4] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, 4th ed., New York: McGraw-Hill, 1989.
- [5] G. Dasgupta and J. Treil, "Maxillo-Facial Frame: Finite Element Shapes," *The Mathematica Journal*, **8**(2), 2001 pp. 235–246.
- [6] P. J. Davis, *Interpolation and Approximation*, New York: Dover Publications, 1963.
- [7] V. L. Rvachev, T. I. Sheiko, V. Shapiro, and I. Tsukanov, "On Completeness of RFM Solution Structures," *Computational Mechanics*, **25**, 2000 pp. 305–316.

About the Authors

Elisabeth Anna Malsch completed her B.S. (1999), M.S. (2000), and Ph.D. (2003) from Columbia University, New York. Currently she is an Alexander von Humboldt fellow, Bonn, Germany. She has been using symbolic computation in her undergraduate, graduate and Ph.D. courses and research focused on engineering mechanics. For details about her recent publications on concave finite elements and boundary elements, see www.civil.columbia.edu/~malsch.

Gautam Dasgupta is a professor of civil engineering and engineering mechanics at Columbia University, New York. His areas of research include computer mathematics, waves in random media, earthquake engineering, and finite and boundary element analysis with *Mathematica*. For details on his recent work, see www.columbia.edu/~gd18.

Elisabeth Anna Malsch

Postdoctoral Researcher
Department of Civil Engineering and Engineering Mechanics
Columbia University
500 West 120th Street
New York, NY 10027
malsch@civil.columbia.edu

Gautam Dasgupta

Professor
Department of Civil Engineering and Engineering Mechanics
Columbia University
500 West 120th Street
New York, NY 10027
dasgupta@columbia.edu