The finite element method in two spatial dimensions

Constructing the spike functions

In a problem with two spatial dimensions, the spike functions we will use as the basis for the Galerkin method are piecewise linear functions made up of several 'facets' glued together. Each facet is constructed by taking a triangle in the plane, lifting one of its corners up to height 1, and then constructing a slice of the plane passing through those three points.

The makePlane function takes a list of three points and constructs the equation of the plane passing through those points.

```
makePlane[{a_, b_, c_}, x_, y_] := Module[{normal},
  normal = Cross[c - a, b - a];
  (-(normal[[1]] (x - a[[1]]) + normal[[2]] (y - a[[2]])) / normal[[3]] + a[[3]])]
```

Each plane we construct will be 'sliced' by restricting its domain to be only the set of points in the triangle that marks out that particular facet. To determine whether or not a point belongs to a particular triangle, we construct three vectors that trace a counter-clockwise path around the triangle. A point is in the interior of the triangle if it lies to the left of each of the sides. The following function constructs a test that determines whether or not the point x lies to the left of the line passing through points a and b.

```
isLeft[x_, a_, b_] := Module[{xE, aE, bE},
  xE = {x[[1]], x[[2]], 0};
  aE = {a[[1]], a[[2]], 0};
  bE = {b[[1]], b[[2]], 0};
  Cross[bE - aE, xE - aE][[3]] > 0]
```

This function tests for points that are to the left or on the line formed by a and b.

```
isLeftOrOn[x_, a_, b_] := Module[{xE, aE, bE},
  xE = {x[[1]], x[[2]], 0};
  aE = {a[[1]], a[[2]], 0};
  bE = {b[[1]], b[[2]], 0};
  Cross[bE - aE, xE - aE][[3]] ≥ 0]
```

Now that we know how to construct the equation of the plane and the test for points that lie in the triangle that will form the domain of the plane function, we can construct a function that will make the part of the piecewise function definition for the facet over the given triangle. The makePart function takes as its parameters a list {a,b,c} of points that make up the vertices of the triangle, a point d which indicates which of the three corners we will be 'lifting' to z=1 (d must be one of the three points in the list {a,b,c}), and the variables x and y.

Note: an important assumption in this function is that the points that make up the triangle have to be given in counter-clockwise order. If they are not, the membership test for the points in the triangle will

not work correctly.

```
makePart[{a_, b_, c_}, d_, x_, y_] := Module[{list, aE, bE, cE},
  list = {a, b, c};
  (* Rotate list around until the first point in the list is d. *)
  While[list[[1]] ≠ d, list = RotateRight[list, 1]];
  (* Extend the three points to three-space. 'Lift' the first point to z = 1. *)
  aE = {list[[1, 1]], list[[1, 2]], 1};
  bE = {list[[2, 1]], list[[2, 2]], 0};
  cE = {list[[3, 1]], list[[3, 2]], 0};
  (* Return the specification for the facet. *)
  {makePlane[{aE, bE, cE}, x, y], And[ isLeftOrOn[{x, y}, list[[1]], list[[2]]],
    isLeft[{x, y}, list[[2]], list[[3]]], isLeft[{x, y}, list[[3]], list[[1]]]]}]
```

To demonstrate how this works in practice, suppose we have a triangle in the x,y plane bounded by the origin and the points {1,0} and {1,1}. If we decide to lift the point {1,1} to {1,1,1}, the specification for the facet that gets formed is

```
makePart[{{0, 0}, {1, 0}, {1, 1}}, {1, 1}, x, y]
```

```
{y, x - y ≥ 0 && y > 0 && 1 - x > 0}
```

We are now in a position to make the definitions for the piecewise linear functions that will be our phi functions. The next function takes a list of all the triangles we have, a point pt that we want to lift and the variables x and y and constructs the list of pairs that Piecewise will need to define the spike function centered at the given point.

```
makeSpikeParts[triangles_, pt_, x_, y_] := Module[{selected},
  (* Select the triangles that contain the point pt. *)
  selected = Select[triangles, MemberQ[#, pt] &];
  (* Make a list of parts, one for each triangular facet. *)
  Map[makePart[#, pt, x, y] &, selected]]
```

```
phi[i_, x_, y_] :=
 Piecewise[makeSpikeParts[triangles, allPoints[[interior[[i]]]], x, y], 0]
```

Setting up the triangles

Here is an initial list of triangles that will define the region in the plane on which we will solve a PDE. The points that make up each triangle are listed in counter-clockwise order.

```
t1 = {{{0, 0}, {2, 0}, {2, 2}}, {{2, 0}, {4, 2}, {2, 2}},
  {{2, 0}, {4, 0}, {4, 2}}, {{4, 0}, {6, 2}, {4, 2}}}
```

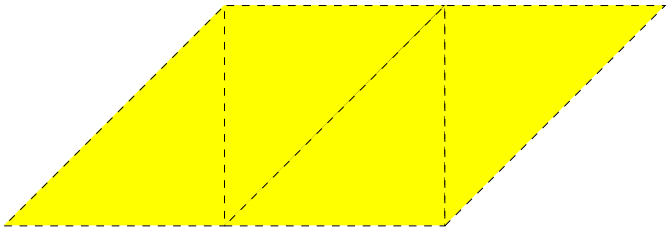
```
{{{0, 0}, {2, 0}, {2, 2}}, {{2, 0}, {4, 2}, {2, 2}},
 {{2, 0}, {4, 0}, {4, 2}}, {{4, 0}, {6, 2}, {4, 2}}}
```

To help us visualize what is going on, here is a function that will construct a graphical representation for a list of triangles.

```
plotTriangles[list_] :=
 Show[Map[Graphics[{EdgeForm[Dashed], Yellow, Polygon[#]}] &, list]]
```

This is a picture of the plane region we will be working with.

```
plotTriangles[t1]
```



Constructing an initial, crude set of triangles to describe the plane region is typical the first step. Usually, this initial set of crude triangles will need to be further sub-divided so we can generate a reasonable set of triangles and sample points at which to define our basis functions phi. A standard technique that gets applied at this point is structured subdivision of the initial triangles into a larger set of smaller triangles. The refine function shown here takes a triangle and returns a list of four smaller triangles constructed by subdividing the original. The refine function preserves the counter-clockwise ordering of the points.

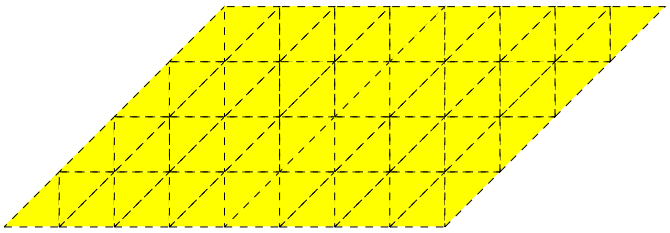
```
refine[{a_, b_, c_}] := {{a, (a + b) / 2, (a + c) / 2}, {(a + b) / 2, b, (b + c) / 2},
  {(a + b) / 2, (b + c) / 2, (a + c) / 2}, {(a + c) / 2, (b + c) / 2, c}}
```

```
refineTriangles[list_] := Flatten[Map[refine, list], 1]
```

Iterating this refinement step leads to a recursive subdivision of the original set of triangles into a larger set of smaller triangles.

```
triangles = Nest[refineTriangles, t1, 2];
```

```
plotTriangles[triangles]
```



This is a reasonable set of triangles to use. The resulting figure has enough interior points to allow us to make a reasonable set of basis functions. At the same time, there are not too many interior points.

Another thing we are going to need is a list of all triangle vertices. Embedded in that list will be all of the interior points at which we want to place phi functions.

```
allPoints = Union[ Flatten[triangles, 1]];
```

A problem with this list of all points is that it contains both interior points and boundary points. We have to explicitly separate out the interior points from the boundary points. There is a function named

ConvexHull found in the Computational Geometry package that will examine a list of points and return the indices of the points that lie on the boundary of the plane figure containing those points.

```
<< ComputationalGeometry`
```

```
boundary = ConvexHull[allPoints]
```

```
{45, 44, 42, 39, 35, 30, 25, 20, 15, 10, 6, 3, 1, 2, 4, 7, 11, 16, 21, 26, 31, 36, 40, 43}
```

The interior points are all the indices not represented in the list of boundary points.

```
interior = Complement[Table[i, {i, 1, Length[allPoints]}], boundary]
```

```
{5, 8, 9, 12, 13, 14, 17, 18, 19, 22, 23, 24, 27, 28, 29, 32, 33, 34, 37, 38, 41}
```

```
Length[interior]
```

```
21
```

To select an interior point, we select an index in the list of interior points and use that as an index into the list of all points constructed above.

```
allPoints[[interior[[1]]]]
```

$$\left\{1, \frac{1}{2}\right\}$$

A typical basis function

We now have all the machinery in place to construct our basis functions. To construct a basis function ϕ_i we

1) Use i as an index into the list of iterior points to select a point.
2) Find all of the triangles that have that point as one of their vertices.
3) Make a facet for each of those triangles that has the selected point lifted to z = 1.
4) Assemble the facets to make a piecewise linear definition for the basis function.

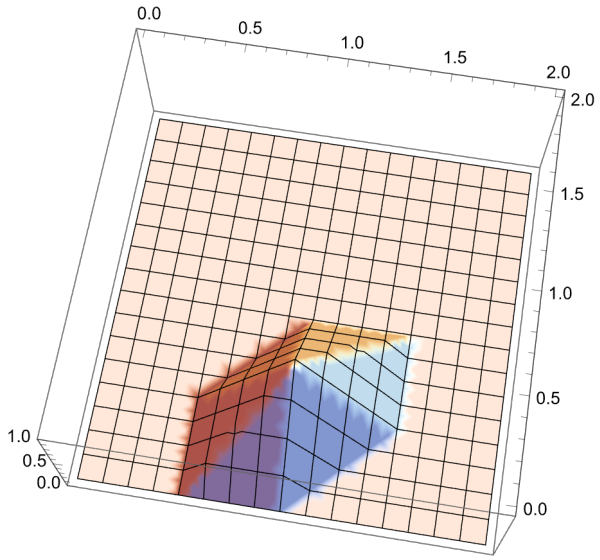
Here is what a typical basis function looks like.

```
phi[1, x, y]
```

$$\begin{cases} 1-2\left(\frac{1}{2}-y\right) & -\frac{1}{4}+\frac{x}{2}-\frac{y}{2}\geq 0 \&\& \frac{y}{2}>0 \&\& \frac{1}{2}-\frac{x}{2}>0 \\ 1-2\,(1-x) & \frac{1}{4}-\frac{y}{2}\geq 0 \&\& -\frac{1}{4}+\frac{x}{2}>0 \&\& \frac{1}{4}-\frac{x}{2}+\frac{y}{2}>0 \\ 1-4\left(\frac{1-x}{2}+\frac{1}{2}\left(-\frac{1}{2}+y\right)\right) & \frac{1}{2}-\frac{x}{2}\geq 0 \&\& \frac{x}{2}-\frac{y}{2}>0 \&\& -\frac{1}{4}+\frac{y}{2}>0 \\ 1-4\left(\frac{1}{2}\,(-1+x)+\frac{1}{2}\left(\frac{1}{2}-y\right)\right) & -\frac{1}{2}+\frac{x}{2}\geq 0 \&\& \frac{1}{2}-\frac{x}{2}+\frac{y}{2}>0 \&\& \frac{1}{4}-\frac{y}{2}>0 \\ 1-2\,(-1+x) & -\frac{1}{4}+\frac{y}{2}\geq 0 \&\& \frac{3}{4}-\frac{x}{2}>0 \&\& -\frac{1}{4}+\frac{x}{2}-\frac{y}{2}>0 \\ 1-2\left(-\frac{1}{2}+y\right) & \frac{1}{4}-\frac{x}{2}+\frac{y}{2}\geq 0 \&\& \frac{1}{2}-\frac{y}{2}>0 \&\& -\frac{1}{2}+\frac{x}{2}>0 \end{cases}$$

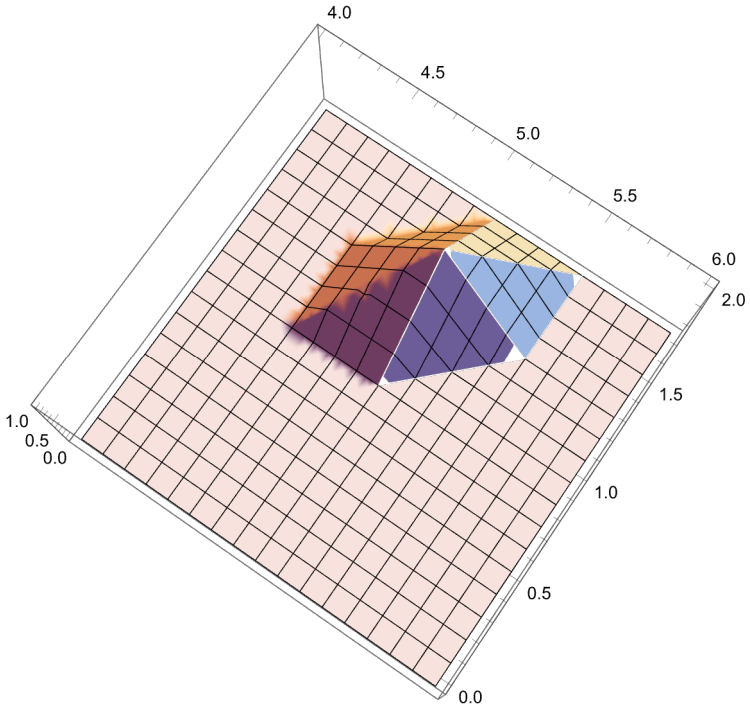
Here is a plot of that basis function clearly showing the six facets that make up that basis function.

```
Plot3D[phi[1, x, y], {x, 0, 2}, {y, 0, 2}, PlotRange → {0, 1}]
```



Here is a picture of the last of the phi functions.

```
Plot3D[phi[21, x, y], {x, 4, 6}, {y, 0, 2}, PlotRange → {0, 1}]
```



Solving a PDE

We are now in the position to solve a PDE. The problem we will solve is Poisson's equation on the rhomboid region we have defined. As discussed in the text, the weak form of Poisson's equation requires us to compute the matrix $K_{i,j} = \int \nabla \phi_i \nabla \phi_j \, d\!l x$ and the vector $F_i = \int \phi_i f \, d\!l x$ and then solve the

matrix equation $K\,u = F$.

To facilitate computing the entries of the K matrix, I introduce the following two function definitions that can be used to pre-compute and remember the gradients of the basis functions.

```
d1Phi[i_] := d1Phi[i] = (D[phi[i, #1, #2], #1]) &
```

```
d2Phi[i_] := d2Phi[i] = (D[phi[i, #1, #2], #2]) &
```

Here now is the definition of the A matrix and the computation of that matrix for our basis functions.

```
A[i_, j_] := NIntegrate[{d1Phi[i][x, y], d2Phi[i][x, y]}.
   {d1Phi[j][x, y], d2Phi[j][x, y]}, {x, 0, 6}, {y, 0, 2}]
```

```
matrixA = Table[A[i, j], {i, 1, Length[interior]}, {j, 1, Length[interior]}];
```

As is typical, the A matrix that results is a banded matrix with mostly 0 entries.

```
MatrixForm[matrixA]
```

$$\begin{pmatrix}
4. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
-1. & 4. & -1. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & -1. & 4. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & -1. & 0. & 4. & -1. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & -1. & -1. & 4. & -1. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & -1. & 4. & 0. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & -1. & 0. & 0. & 4. & -1. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & -1. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & 0. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & 0. & 4. & -1. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & -1. & 0. & -1. & 0. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & 0. & 0. & -1. & 0. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & 0. & 4. & -1. & 0. & -1. & 0. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & -1. & 0. & -1. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & 0. & 0. & -1. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & 0. & 4. & -1. & 0. & 0. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & -1. & -1. & 0 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & -1. & 4. & 0. & - \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & 0. & 4. & - \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -1. & -1. & 4 \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & -
\end{pmatrix}$$

Here is the function we will use for the right-hand side of the Poisson equation.

```
f[x_, y_] := x y
```

```
vectorB[i_] := NIntegrate[phi[i, x, y] × f[x, y], {x, 0, 6}, {y, 0, 2}]
```

```
b = Table[vectorB[i], {i, 1, Length[interior]}]
```

```
{0.130208, 0.192708, 0.380208, 0.255208, 0.505208, 0.755208,
 0.317708, 0.630208, 0.942708, 0.380208, 0.755208, 1.13021, 0.442708,
 0.880208, 1.31771, 0.505208, 1.00521, 1.50521, 1.13021, 1.69271, 1.88021}
```

Finally, we solve the system of equations and plot the approximate solution.

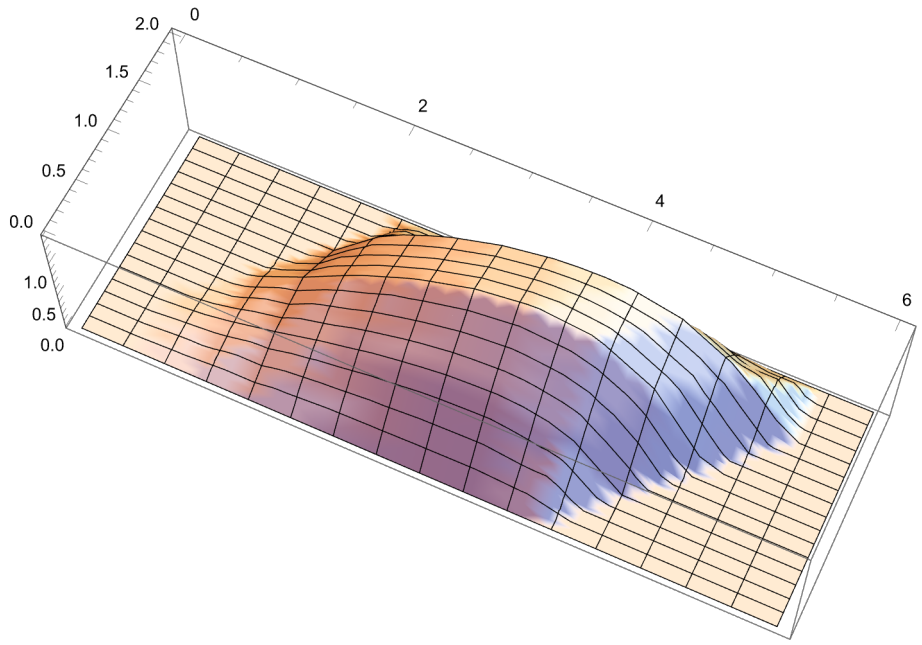
```
c = LinearSolve[matrixA, b]
```

```
{0.104463, 0.287643, 0.358183, 0.495217, 0.76488, 0.618687, 0.673137,
 1.08222, 0.954659, 0.797399, 1.30601, 1.17502, 0.83024, 1.4142,
 1.30918, 0.666652, 1.33116, 1.32981, 0.908764, 1.17369, 0.763474}
```

```
soln[x_, y_] := Sum[phi[i, x, y] × c[[i]], {i, 1, Length[interior]}]
```

The plot matches our intuition for what the solution to this PDE should look like.

```
Plot3D[soln[x, y], {x, 0, 6}, {y, 0, 2}, BoxRatios → Automatic]
```



The elevation of various parts of the figure is consistent both with the Dirichlet boundary condition and the shape of the external forcing function:

```
Plot3D[f[x, y], {x, 0, 6}, {y, 0, 2}, BoxRatios → Automatic]
```

