Vinicio Haro, STAT 786 FEM

In[16]:=
```
(*Segment One: Importing the data and drawing the region*)

nodes = Import["/Users/Kathrynharo/Desktop/ChnlNodes(2).Dat"];
elements = Import["/Users/Kathrynharo/Desktop/ChnlElems(1).dat"];

i = 1;
nodeslist = {};

(*Iteration count is 1. The following code fragment will construct 188 nodes and
 the AppendTo function will append y to a value of x and then reset [x, y]*)

While[i ≤ Length[nodes],
  nodeslist = AppendTo[nodeslist, {nodes[[i]][[2]], nodes[[i]][[3]]}];
  i++];

(*155 Elements are being assembled here. Since each element is a square,
the i's go in the sequence 2, 3, 4, 5, 2, 3, 4, 5...*)

 partition={};
 i=1;
 While[i≤Length[elements],
  AppendTo[partition,
     Line[{nodeslist[[elements[[i]][[2]]]],
       nodeslist[[elements[[i]][[3]]]],
       nodeslist[[elements[[i]][[4]]]],
       nodeslist[[elements[[i]][[5]]]],
       nodeslist[[elements[[i]][[2]]]]
      }]]
    i++;
 ];


 (*Segment 2: Define Lagrange polynomials and construct the K Matrix*)

 N1 = (γ-x)/(γ-α) (δ-y)/(δ-β);
 N2 = (x-α)/(γ-α) (δ-y)/(δ-β);
 N3=(x-α)/(γ-α) (y-β)/(δ-β);
 N4=(γ-x)/(γ-α) (y-β)/(δ-β);

 polynomialset = {N1, N2, N3, N4};

 (*Background information required to accuratly construct K Matrix
```

$\nabla^2\mu$=0 is the Laplace Equation however,
we want to take it's weak form.Recall Greene's theorem:

$\int_\Omega \nabla\mu\nabla\psi_i = \int_\Omega \frac{\partial\mu}{\partial\mu}\psi_i$

Now we can write the weak form of the

Laplace: $\int_\Omega \left(\nabla^2\right)_n \psi_i = 0$ and ψ is a join function of the polynomialsLet $\mu = \sum \alpha_j \psi_j$

$\sum \alpha_i \int \nabla\psi_i \nabla\psi_j = \int \frac{\partial \mu}{\partial \mu} \psi_i$

Hence a linear system of equations : $K\alpha = R$

*)

```
(* With the above bakground information in mind,
we can now define what we want our K vector to be. The strategy here
 will be to write three nested loops: Loop on elements to get the alpha,
beta, gamma, and delta values then loop on polynomials to make
 them local then taking the gradient, then another loop on
 polynomials and take the gradint. Once the gradient is obtained,
I need to compute ∫∇NiNj = δ_ij which will be done using numerical integration*)

KMatrix =ConstantArray[0, {Length[nodeslist],Length[nodeslist]}];

(*IdentityMatrix[Length[nodeslist]]-IdentityMatrix[Length[nodes]];*)
(* Loop on elements *)

i=1;
While[i≤Length[elements],
 elements1=elements[[i]];
 alphavalue=nodes[[elements1[[2]],2]];
 betavalue=nodes[[elements1[[2]],3]];
 gammavalue=nodes[[elements1[[4]],2]];
 deltavalue=nodes[[elements1[[4]],3]];

 (*Frst loop on polynomials. A few lines up,
 the polynomials were defined in terms of alpha, beta, gamma, delta, x, and y*)
 (*The way my elements are set up,
 gave me complications for building k causing me to take the jth-
  1 and the kth-1 entries instead of the kth and jth *)
 (*Creating a row matrix*)
 j=1;
 While[j≤Length[polynomialset],
  alternatepolynomials[α_, β_, γ_, δ_, x_, y_] =polynomialset[[j]];

  (*Now we need to make them local*)

  localalternatepolynomials[x_,y_]=
   alternatepolynomials[alphavalue, betavalue, gammavalue, deltavalue, x, y];

  (*I will now attempt to take the gradient of the polynomials using
    the Grad function and the D function which takes the partial
    derivative. Recall the basic definition of gradient ∇=(∂x,∂y,∂z....)*)

  Gradient1[x_, y_]=
   {D[localalternatepolynomials[x,y],x],D[localalternatepolynomials[x,y],y]};
  jmatrix=elements1[[ j+1]];

  (*Now we move on to the second loop on polynomials. It will almost be
```

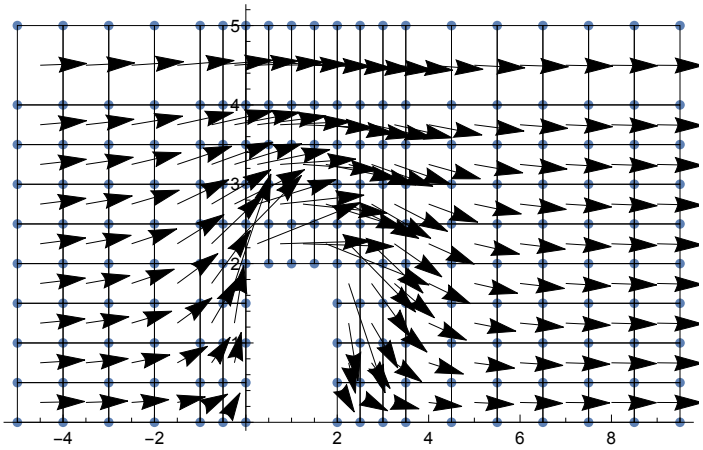
```
   identical to jmatrix however, we now seek to build some column matrix soon
   to be called ymatrix. I cannot use kmatrix because k is already being used*)

  (*creating a column matrix*)
  k=1;
  While[k≤Length[polynomialset],
   alternatepolynomials1[α_, β_, γ_, δ_, x_, y_] =polynomialset[[k]];
   localalternatepolynomials1[x_,y_]=
    alternatepolynomials1[alphavalue, betavalue, gammavalue, deltavalue, x, y];
   Gradient2[x_, y_]={D[localalternatepolynomials1[x,y],x],
     D[localalternatepolynomials1[x,y],y]};
   ymatrix=elements1[[ k+1]];

   (*Now with three nested loops written,
   we can perform 4 point gaussian quadrature in order to compute the
    integrals much faster than using regular integration for 155 elements*)

   integrals=NIntegrate[Gradient1[x,y].Gradient2[x,y],
     {x, alphavalue, gammavalue},{y,betavalue, deltavalue}];
   (*Lets build the K matrix*)

   KMatrix[[jmatrix,ymatrix]]=KMatrix[[jmatrix, ymatrix]]+integrals;
   k++
  ];
  j++
 ];
 i++
];

(*Part 3: Construct the R vector and proceed to post processing*)
(*Structure is similar to K matrix except I am only looking
 at the right and left edge. Everywhere else, will be zero*)

(*Some bakground required for constructing the R vector: R_i^e=∫_Γ ∂ψ_e/∂n N_i^e=-∫_Γ N_i^e  *)

Rvector=ConstantArray[0,188];

(*My first loop will be for the ledt
 side of the geometry so I need nodes 1 and 4 first*)

i=1;
While[i≤9,
 elements1=elements[[i]];
 alphavalue=nodes[[elements1[[2]],2]];
 betavalue=nodes[[elements1[[2]],3]];
 gammavalue=nodes[[elements1[[4]],2]];
 deltavalue=nodes[[elements1[[4]],3]];

 (*I'm allocating my polynomials to poly1 and poly4
  pertaining to the nodes that I am interested in for quicker access*)

 poly1[α_, β_, γ_, δ_, x_, y_] = polynomialset[[1]];

 Rvector[[elements[[i]][[2]]]]=Rvector[[elements[[i]][[2]]]]+
```

```
    NIntegrate[-1*poly1[alphavalue,betavalue,gammavalue,deltavalue,alphavalue,y],
     {y,betavalue,deltavalue}];

 poly4[α_, β_, γ_, δ_, x_, y_] = polynomialset[[4]];

 Rvector[[elements[[i]][[5]]]]=Rvector[[elements[[i]][[5]]]]+
    NIntegrate[-1*poly1[alphavalue,betavalue,gammavalue,deltavalue,alphavalue,y],
     {y,betavalue,deltavalue}];
 i++
];

(*Using a similar structure and argument above, I need the right side of
 the geometry pertaining to nodes 2 and 3 allocated to poly2 and poly3*)

i=147;
While[i≤155,
 elements1=elements[[i]];
 alphavalue=nodes[[elements1[[2]],2]];
 betavalue=nodes[[elements1[[2]],3]];
 gammavalue=nodes[[elements1[[4]],2]];
 deltavalue=nodes[[elements1[[4]],3]];

 poly2[α_, β_, γ_, δ_, x_, y_] = polynomialset[[2]];

 Rvector[[elements[[i]][[3]]]]=Rvector[[elements[[i]][[3]]]]+
    NIntegrate[1*poly1[alphavalue,betavalue,gammavalue,deltavalue,alphavalue,y],
     {y,betavalue,deltavalue}];

 poly3[α_, β_, γ_, δ_, x_, y_] = polynomialset[[3]];

 Rvector[[elements[[i]][[4]]]]=Rvector[[elements[[i]][[4]]]]+
    NIntegrate[1*poly1[alphavalue,betavalue,gammavalue,deltavalue,alphavalue,y],
     {y,betavalue,deltavalue}];
 i++
];

(*The R vector and the K matrix are now
 complete. I need to make a slight adjustment to account
 for the obstruction in the region which is at element 61*)
```

Out[21]= Null

```
zerovector = ConstantArray[0, 188];
plots = ConstantArray[0, Length[elements]];

zerovector[[61]] = 0;
KMatrix[[61]] = zerovector;
Rvector[[61]] = 0;

(*Everything is now ready to apply the linear solve function*)


answer = LinearSolve[KMatrix, Rvector];

(*Part 4: Post Processing. I want to now
  draw the region with the corresponding vector field*)

(*The strategy here to get the vector field will be to show the flow field
 at the centroid of each element and the centroid will simply be denoted
 as Cx and Cy. We recall that each element has nodes {Ni, N2, N3, N4}. The
 speed of the flow is represented as the length of the vector arrows*)

i = 1;
While[i ≤ Length[elements],

  elements1 = elements[[i]];
  alphavalue = nodes[[elements1[[2]], 2]];
  betavalue = nodes[[elements1[[2]], 3]];
  gammavalue = nodes[[elements1[[4]], 2]];
  deltavalue = nodes[[elements1[[4]], 3]];

  cx = (alphavalue + gammavalue) / 2;
  cy = (betavalue + deltavalue) / 2;

  field[x_, y_] := answer[[elements[[i]][[2]]]] *
     poly1[alphavalue, betavalue, gammavalue, deltavalue, x, y] +
    answer[[elements[[i]][[3]]]] * poly2[alphavalue, betavalue, gammavalue,
      deltavalue, x, y] + answer[[elements[[i]][[4]]]] * poly3[alphavalue,
      betavalue, gammavalue, deltavalue, x, y] + answer[[elements[[i]][[5]]]] *
     poly4[alphavalue, betavalue, gammavalue, deltavalue, x, y];

  flow = Grad[field[x, y], {x, y}] /. {x → cx, y → cy};

  plots[[i]] = Arrow[{{cx, cy}, {cx, cy} + flow}];
  i++
 ];

Show[ListPlot[nodeslist], Graphics[partition], Graphics[{plots}]]
```

Out[30]=



In[31]:=
```
(*This is the farthest I could get. I need more
 time to achieve the streamlines by using forward Euler*)
```