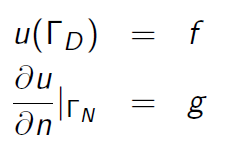
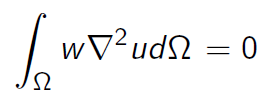
*1. Starting from the strong form of the partial differential equation, derive the relevant weak form that will be used as the basis of the finite element method.*

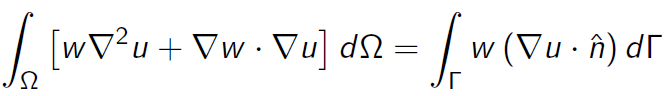
The strong form of the potential flow equation is given by (1) with boundary conditions (2):

(1)  
  (2)

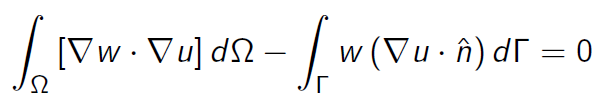
Multiplying by a weighting function and integrating over the domain yields (3):

 (3)

Using Green’s theorem and substitution of similar problem variables yields (4):

 (4)

To satisfy the potential equation, the term in (4) is zero; rearranging yields (5), the weak form of (1):

 (5)

*2. Show how the weak form that you have derived represents the minimization of the kinetic energy in the potential flow representation.*

For a non-dimensionalized unit mass, kinetic energy is given by (6), and for the whole domain by (7):

(6)

(7)

A perturbation to the potential flow *w*, adds additional kinetic energy to the domain, calculated in (8), (9), and (10):

(8)

(9)

(10)

The kinetic energy due to the perturbative field must be greater than zero, unless the perturbation is trivially zero. No such explicit imposition is placed on the term. Therefore minimizing the kinetic energy in the domain entails finding the configuration that sets , precisely the approach taken by FEM in (5).

*3. You will use linear-hat nodal basis functions to represent the solution and the test functions in your solver. Describe/draw how these linear hat basis functions look in two dimensions for:*

*(a) For a node i in the triangulation, show the "tent" basis function (nodal hat basis function)*

*(b) For an element k, show the three linear basis functions on a triangular element.*

Figure 1 represents graphically the nodal linear hat functions: Green indicates lines in the x-y plane, red represents vertical lines (parallel to the z-axis), and blue indicates lines in a non-specific direction not in the x-y plane. All red lines are 1 unit tall, all nodes are on the x-y plane. All three elements in b) are the same identical element, with each basis function indicated for each node. The z-value of the function of a point on the plane defined by the two blue vectors indicates the value of the basis function from at a point in the x-y plane correlated with the projection of that point onto the x-y plane.

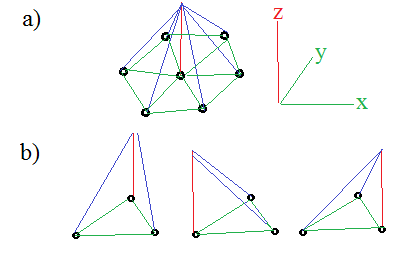
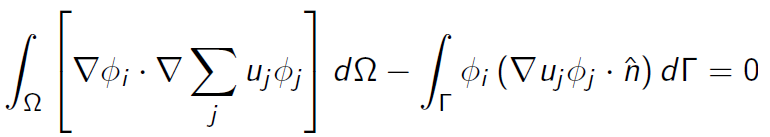


Figure 1: Graphical representation of linear hat functions

*4. Write the discrete form of the weak form of the equation for a weighting function/node i in the domain.*

Equation (5) can be discretized in general by substituting in the appropriate functions for the weighting function and test function for the potential by (11):

 (11)

If the basis function is described as a plane as indicated in Figure 1, its equation can be parameterized as ax+by+c = z. The divergence of this function is given by (12):

(12)

Therefore using (11) and (12) an individual node would be given by (13) for a node i and basis function from j:

(13)

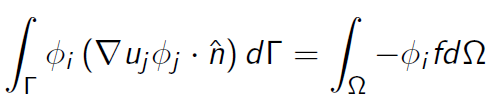
*5. Write the discrete form of the weak form of the equation for a node i on the boundary of the domain where you have Dirichlet BCs.*

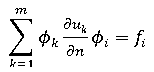
As given in the first equation of (1) the Dirichlet BC is an explicit setting of the value of *u* at that boundary node. This simply involves insertion of an explicit ‘1’ at Ai,i  and zeroing the rest of that row. This implicitly gives the relation as given in (14):

(14)

*6. Write the discrete form of the weak form of the equation for a node i on the boundary of the domain where you have Neumann BCs.*

At the boundary the discrete weak form from (11) with a BC ‘f’, is given by (15):

 (15)

where the weighting basis function for the BC on the boundary is 1. is precisely the Neumann BC. Examining at a particular node *i,* with‘m’ nodes connected by elements locally numbered 1-m (mappable to their actual global node numbers), using (15) and applying the relation in (13) yields (16):  (16)

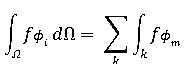
This is satisfied naturally by the FEM with no modification of the ‘A’ matrix required other than to insert the desired BC value at.

*7. Derive the elemental matrices that will be used to build the overall A-matrix (stiffness matrix). Also derive the elemental load vectors that will be used to determine the RHS. Include how you will calculate the gradient values using the equation for a plane surface.*

Using the discrete form from (11) and applying the relation between the basis functions in (13) at an element with local node numbers 1-3 (which correspond to mappable global node numbers) yields the elemental matrix (17):

(17)

The RHS can be isolated from (15) and with some modifications can be adapted to an elemental form (18), where m indicates indexing by local node number:

 (18)

Modifying (18) gives the matrix value at a single node, using the area of a tetrahedron gives (19):

(19)

The basis function is a plane uniquely defined by the three points on its surface. This creates a system of equations that can be solved simultaneously. The solution for the basis function for local node 1 in matrix form is (20):

(20)

A set of equations exists for each of the three basis functions in the element. Each one can be solved for the ‘a’ and ‘b’ coefficients and used to calculate (17) with the gradients calculated using (13).

*8. Describe the Stamping procedure that will be used to construct the global matrix using the*

*elemental matrices. You may use both words and example matrices.*

The local elemental node numbers correspond to global node numbers in the ‘A’ matrix. Therefore the locally produced elemental matrix from (17) must be mapped to the correct ‘A’ rows. Rows of order are formed with their respective columns dictated by .

For instance suppose local i=1 node corresponds to 12, i=2 to 14, and i=3 to 27. Then the second row of (17) would be inserted into A(14,12), A(14,14), and A(14,27) moving column-wise from left to right. Likewise the other two rows would be inserted into the correct row dictated by .

One important consequence of this elemental stamping procedure is that if the surface isn’t homogenously tessellated, then some rows in the A matrix will contain more entries then others if they are more connected to other local nodes as all of them will have stamped their respective entries into that nodes row.

*9. For each boundary condition, describe the implementation in your FEM code.*

For Dirichlet BCs the row corresponding to the node where the condition is to be enforced will be zeroed out except for the column of the same rank as the nodes row; This entry will be set to ‘1’ to explicitly select the corresponding *u* term of the same row rank. The *f* value of that row rank sets the enforced nodal value.

For Neumann BCs the equation is naturally satisfied, so the A matrix requires no further modification. All that remains is to set *f* to the desired gradient value.

*10. Write pseudo code describing the finite element method implementation that you will use.*

Generate NACA airfoil shape and nodes.

Generate the domain limits and node spacing

Perform a triangular meshing of the surface

for: each element

calculate connected node numbers and x-y locations

calculate triangular bounded area

calculate planar coefficients for elemental basis functions

calculate value of gradient dot products and assemble elemental matrix

calculate elemental forcing values

stamp elemental stiffness and forcing matrices into ‘A’ and ‘F’ matrices

end

Apply Neumann BCs

Apply Dirichlet BCs

Apply airfoil BCs

Solve matrices for ‘u’ matrix

Plot mesh, velocity, and potential distributions

*11. Add the elemental matrices and boundary conditions to the provided pseudo code and complete the FEM solver so that it works (currently the solver has some random numbers in places to allow it to "work"). You may optionally develop your own two-dimensional FEM solver. If you choose to use the Matlab version provided, please read through and understand the comments and structure. If you choose to develop your own solver, please use the geometry definition tool provided.*

The skeleton code provided was modified with the proper BC, stiffness, and forcing relations. Additionally some modifications were made to the discrete geometry generating code to output all relevant boundaries. The code is presented in Appendix A.

*12. Solve the potential flow problem and plot the scalar potential using a minimum grid refinement value of 3.*

Using the code from Appendix A, the potential flow was calculate for an airfoil (NACA 0045) with Dirichlet BCs equal to the x value along the left and right sides, and Neumann BCs applied along the top, bottom, and airfoil surface. Figure 2 plots the results of the calculated flow potential.

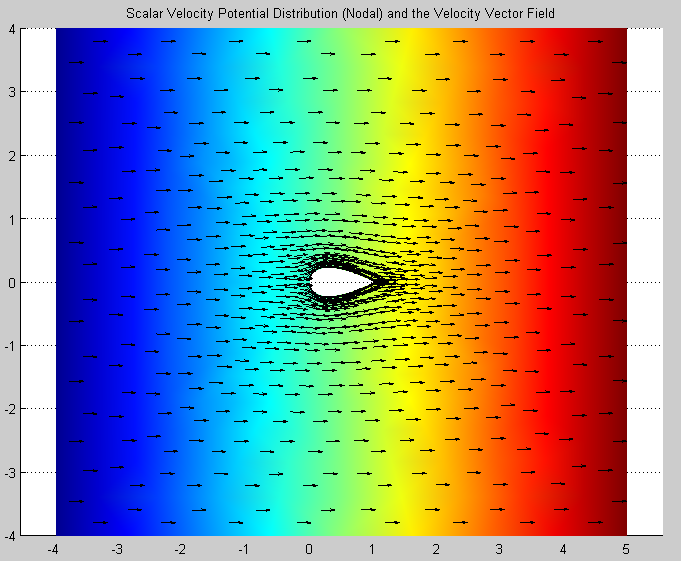


Figure 2: Potential flow around NACA 0045 airfoil calculated by a FEM approach

*13. Show how the value of your scalar potential solution at the front of the object (location 0,0) converges with grid refinement.*

The value of the flow potential at the front of the airfoil x=y=0 was calculated for a range of refinement sizes up to ref=8. The inf-norm of the error for the potential values for the less refined runs were plotted on a log-log plot versus node number along the longitudinal axis that passes through the center of the airfoil. If we assume that the convergence follows a polynomial relationship then the convergence appears to be 3rd order. Figure 3 suggests that the convergence rate may actually accelerate with increasing node number.

Figure 3: Convergence of Inf-Norm error for stagnation point at front of airfoil

**Appendix A: Source Code**

Note: several \*.m scripts that were left unmodified were left out for brevity.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Josh Bevan 2013

%Based on code provided by Prof. WIllis, Umass Lowell

%

%Num. Methods for PDEs, 22.520

%Hw 4

clc

close all;

clear all;

%=========================================================================

% Start by defining your domain: Meshing parameters (done for you)

%=========================================================================

% The following parameters are used for defining the mesh of the object

%

% Shape: This selects an ellispoid or a NACA-4 Digit airfoil

% -- Ellipsoid: Set the shape to a single value (the aspect ratio of

% the ellipsoid shape. eg: Shape = [1];

% -- NACA-4-Digit airfoil: Set the shape parameter a 2-valued entry,

% eg: Shape = [1 0012], where the second value is the 4-digit reference

% for the NACA airfoil.

%

% DomainSize: This is a basic parameter that defines how far the domain

% should extend (approx.) from your shape. A value of 4 should be

% sufficient, but you can play around with this.

%

% ref: This is the refinement of the mesh. The higher the value, the more

% elements and vertices you will have.

%

% powerRef: This is the mesh refinment control. The larger the number, the

% more refined the mesh is near the object. A value of 1 produces a uniform

% refiment. A value of 1.75-2.0 should work well for most of your problems.

%

%=========================================================================

Shape = 1%[1 0045];

DomainSize = 4;

ref = 3;

powerRef = 2;

[TRI, Nodes, Top, Bottom, Left, Right, InnerBoundary] = getDiscreteGeometry(Shape, DomainSize, ref, powerRef);

% plot the triangulation

trimesh(TRI, Nodes(:,1),Nodes(:,2))

axis equal

pause(0.1)

% Determine the number of nodes and elements in the domain

NNodes = length(Nodes);

NElem = length(TRI);

% Initialize the A-matrix and RHS to zero

A = spalloc(NNodes, NNodes, 12\*NNodes);

F = zeros(NNodes, 1);

F0 = 0;

% cycle through the elements to build the A-matrix and f-vector

for(ie=1:NElem)

element\_number = ie;

N1 = TRI(ie,1);

N2 = TRI(ie,2);

N3 = TRI(ie,3);

X1 = Nodes(N1,1);

X2 = Nodes(N2,1);

X3 = Nodes(N3,1);

Y1 = Nodes(N1,2);

Y2 = Nodes(N2,2);

Y3 = Nodes(N3,2);

TriArea = calculateArea(X1, X2, X3, Y1, Y2, Y3);

%Calculate gradient coeffs

Plane = [X1 Y1 1;...

X2 Y2 1;...

X3 Y3 1];

Gradient(:,1) = Plane\[1;0;0];

Gradient(:,2) = Plane\[0;1;0];

Gradient(:,3) = Plane\[0;0;1];

Gradient(3,:) = 0; %Remove unwanted c coeff

gN1gN1 = dot(Gradient(:,1),Gradient(:,1));

gN2gN2 = dot(Gradient(:,2),Gradient(:,2));

gN3gN3 = dot(Gradient(:,3),Gradient(:,3));

gN1gN2 = dot(Gradient(:,1),Gradient(:,2));

gN1gN3 = dot(Gradient(:,1),Gradient(:,3));

gN2gN3 = dot(Gradient(:,2),Gradient(:,3));

A\_elemental = TriArea\*[gN1gN1 gN1gN2 gN1gN3;...

gN1gN2 gN2gN2 gN2gN3;...

gN1gN3 gN2gN3 gN3gN3];

F\_elemental = (1/3)\*TriArea\*[F0 F0 F0]; %If the f is non-constant this would read in the relevant nodal forces

%Note that manually poking values to A and F are slow, it would be better to

%pre-generate the diagonals and create the sparse matrix with spdiags()

A(N1, N1) = A(N1, N1) + A\_elemental(1,1);

A(N2, N1) = A(N2, N1) + A\_elemental(2,1);

A(N3, N1) = A(N3, N1) + A\_elemental(3,1);

A(N1, N2) = A(N1, N2) + A\_elemental(1,2);

A(N2, N2) = A(N2, N2) + A\_elemental(2,2);

A(N3, N2) = A(N3, N2) + A\_elemental(3,2);

A(N1, N3) = A(N1, N3) + A\_elemental(1,3);

A(N2, N3) = A(N2, N3) + A\_elemental(2,3);

A(N3, N3) = A(N3, N3) + A\_elemental(3,3);

F(N1) = F(N1) + F\_elemental(1);

F(N2) = F(N2) + F\_elemental(2);

F(N3) = F(N3) + F\_elemental(3);

end

% Boundary conditions

%Neumann BCs on Top/Bottom/Airfoil

for i=1:length(Top)

F(Top(i)) = 0;

end

for i=1:length(Bottom)

F(Bottom(i)) = 0;

end

%Airfoil boundary

for i=1:length(InnerBoundary)

searcher = InnerBoundary(i,1);

found = find(abs(Nodes(:,1)-searcher)<1e-10);

F(found) = 0;

end

%Dirichlet BCs on Left/Right

for i=1:length(Left)

%Left

A(Left(i),:) = 0;

A(Left(i),Left(i)) = 1;

F(Left(i)) = Nodes(Left(i),1);

%Right

A(Right(i),:) = 0;

A(Right(i),Right(i)) = 1;

F(Right(i)) = Nodes(Right(i),1);

end

% Solve matrix

Sol = A\F;

% Post Processing calculations: Don't Mess with what works :-)

% Gradient calculation

for(ie=1:NElem)

element\_number = ie;

N1 = TRI(ie,1);

N2 = TRI(ie,2);

N3 = TRI(ie,3);

X1 = Nodes(N1,1);

X2 = Nodes(N2,1);

X3 = Nodes(N3,1);

Y1 = Nodes(N1,2);

Y2 = Nodes(N2,2);

Y3 = Nodes(N3,2);

S1 = Sol(N1);

S2 = Sol(N2);

S3 = Sol(N3);

C = [1 X1 Y1; 1 X2 Y2; 1 X3 Y3]\[eye(3)];

Gradient\_IE(ie,:) = [(S1\*C(2,1)+S2\*C(2,2)+S3\*C(2,3)), ...

(S1\*C(3,1)+S2\*C(3,2)+S3\*C(3,3))];

Centroid(ie,:) = [(X1+X2+X3)/3, (Y1+Y2+Y3)/3];

end

% Figure 1

figure

trisurf(TRI, Nodes(:,1), Nodes(:,2), Sol-500) % The 500 is random right now

hold on

trisurf(TRI, Nodes(:,1),-Nodes(:,2),Sol-500)

quiver(Centroid(:,1), Centroid(:,2), Gradient\_IE(:,1), Gradient\_IE(:,2),.75,'k')

quiver(Centroid(:,1), -Centroid(:,2), Gradient\_IE(:,1), -Gradient\_IE(:,2),.75,'k')

shading interp

title('Scalar Velocity Potential Distribution (Nodal) and the Velocity Vector Field')

view([0 0 1])

axis equal

% Figure 2

figure

Vel = ((Gradient\_IE(:,1).^2 + Gradient\_IE(:,2).^2).^(.5))';

a = trisurf(TRI, Nodes(:,1), Nodes(:,2), Nodes(:,2)\*0, Vel);

hold on

set(a,'edgealpha',0)

trisurf(TRI, Nodes(:,1), -Nodes(:,2), -Nodes(:,2)\*0, Vel)

title('Velocity Distribution (Centroidal)')

view([0 0 1])

axis equal