Numerical Simulation of Potential Flow using the Finite Element Method

Cynthia Vanessa Flores (CSUN),
Eduardo Rogério Fávaro(Ibilce-Unesp)
Gilberto Pereira Sassi (ICMC-USP)
Leonardo Koller Sacht(UFSC)
Paulo Liboni (CCET - UFSCar)
Yeranuhi Semerdjian (CSUN)

Summer 2007

Partially supported by NSF grant OISE-0526008.

1 One Dimensional Model Problem

Our model problem consists of a one-dimensional, “two-point” boundary-value problem represented by the following linear ordinary differential equation of second order:

\[-u'' + u = x, \quad 0 < x < 1 \]
\[u(0) = 0, \quad u(1) = 0 \]

(1)

It is easy to verify that the exact analytical solution to (1) is \( u(x) = x - \left( \frac{\sinh x}{\sinh 1} \right) \).

We can use the Finite Element Method to produce an approximation of the solution by forming a variational statement to the classical boundary value problem. One way to produce a weaker variational statement is to choose a set of test and trial functions, \( H_0^1 \). This set consists of functions that vanish at the boundaries and whose first derivatives are square-integrable. Hence, the variational statement takes the following form:

\[ \int_0^1 (-u'' + u)v \, dx = \int_0^1 xv \, dx, \]  

(2)
where \( u, v \in H^1_0 \) are the trial and test functions respectively. The solution to (2) will ensure that (1) and the boundary conditions are satisfied in a “weighted average” sense. Manipulating (2), we arrive at

\[
\int_0^1 (u'v' + uv - xv) \, dx = 0 \quad \forall v \in H^1_0,
\]

\[ (3) \]

1.1 Galerkin Approximations

For practical purposes, we project \( H^1_0 \) to an N-dimensional subspace, \( H^N_0 \). In order to approximate the solution of our model problem, we also project the test and trial functions using a suitable basis \( \{ \phi_i \} \), \( i = 1, ..., N \) for \( H^N_0 \). Expressing the test and trial functions as linear combinations of the basis functions,

\[
u_N(x) = \sum_{i=1}^{N} \beta_i \phi_i(x)
\]

\[ (4) \]

\[
u_N(x) = \sum_{i=1}^{N} \beta_i \phi_i(x)
\]

\[ (5) \]

we arrive at the following system of linear equations:

\[
\sum_{j=1}^{N} K_{ij} \alpha_j = F_i, \quad i = 1, 2, ..., N,
\]

\[ (6) \]

where

\[
K_{ij} = \int_0^1 \left[ \phi'_i(x) \phi'_j(x) + \phi_i(x) \phi_j(x) \right] \, dx
\]

\[ (7) \]

and

\[
F_i = \int_0^1 x \phi_i \, dx.
\]

\[ (8) \]

The matrix \( K = [K_{ij}] \) is called the stiffness matrix, and the vector \( F = [F_i] \) is called the load vector. Since the stiffness matrix is nonsingular, the approximate solution on the space \( H^N_0 \) is uniquely determined by the system (6).

The Finite Element Method involves dividing the domain, \( \Omega \), into \( N - 1 \) subdomains \( \Omega_e \) called elements whose endpoints are nodal points. In the one-dimensional case, this creates \( N \) nodal points. The set of elements and nodal points make up the finite
element mesh. This allows us to approximate the solution over each element with relative ease. Generally, nodal points are assigned to points of discontinuity occurring in the domain. Since there are no discontinuities in the domain of our model problem, it is convenient to divide $\Omega$ into elements of equal length $h = \frac{1}{N}$.

It is important to note that the choice of basis functions will facilitate the process of writing the code. We impose two conditions on the basis functions $\phi_i$. First, the $\phi_i$ should be generated by simple functions defined piecewise, element by element. Second, the $\phi_i$ should belong to $H^1_0$. We chose linear $\phi_i$ such that:

$$
\phi_i(x_j) = \begin{cases} 
1 & i = j \\
0 & i \neq j 
\end{cases} 
$$

where $x_j$ is a nodal point and $i, j \in \{1, \ldots, N\}$. The piecewise construction of the $\phi_i$ naturally defines these local basis functions over each element:

$$
\psi^e_A(\xi) = 1 - \frac{\xi}{h} \quad \xi = x - x_A 
$$

$$
\psi^e_B(\xi) = \frac{\xi}{h} \quad \xi = x_B - x 
$$

where $x_A$ and $x_B$ are the left and right nodal points of element $\Omega_e$ respectively. Hence, we can define a local element stiffness matrix and local element load vector expressed by

$$
k^e_{ij} = \int_{\Omega_e} (\psi'_i \psi'_j + \psi_i \psi_j) \, d\xi 
$$

$$
f^e_i = \int_{\Omega_e} (x_A + \xi) \psi_i \, d\xi 
$$

Now, we can construct the Stiffness Matrix $K$ and Load Vector $F$ using

$$
K_{ij} = \sum_{e=1}^{N} k^e_{ij} 
$$

$$
F_i = \sum_{e=1}^{N} f^e_i 
$$

The theory behind the model problem can be easily extended to a generalized boundary value problem of the form

$$
a_0(x)u''(x) + a_1(x)u'(x) + a_2(x)u(x) = f(x) 
$$
1.2 Code Implementation

Once our team was comfortable with the theory behind the one-dimensional case, we began coding. To start off, we were provided with "black-boxes" (unfinished pieces of C++ code). In order to calculate the integrals (7) and (8) we wrote a numerical integration routine called

double basicintegration(double a, double b);

over the interval \([a,b]\). To numerically compute the integrals, we used the Gaussian Quadrate Method with ten points. To compute the global stiffness matrix, we used an overlapping function given in one of the black boxes. The reason for the need to overlap arose from the way we interpreted the theory; we used two local basis functions which created a \(2 \times 2\) local stiffness matrix. Thus to calculate the global stiffness matrix, we needed to combine the contributions from each local stiffness matrix. We built the local load vectors over every element using (8), and took the sum to construct the global load vector.

Using a black box that implements the Gradient-Conjugate Method, we computed the solution to the linear system (6). We were able to use this method because \([K_{ij}]\) satisfies the hypothesis. Here is a simple outline of our code:

- Define equation to be solved
- Ask for initial point, end point, and number of nodes
- Calculate local stiffness matrix and local load vector
- Assemble the global stiffness matrix and global load vector
- Solve system (6)
- print output

Any plotting program can be used to view the graphical results of the approximate solution. The solution of the model problem as approximated from our code can be compared to the analytic solution in the following figures:
We observe that as the number of nodes increases, our approximation becomes smoother.

An important technical note: our program uses Unix specific memory allocation methods. Thus, the program crashes when used on a Windows Operating system.

2 Two Dimensional Model Problem

Our two dimensional model problem consists of the following differential equation and boundary conditions defined on a rectangular domain:

\[
\begin{align*}
\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2} & = f \equiv 0 \\
\frac{du}{dy} & = -15 \quad \text{for } y = 4 \\
\frac{du}{dy} & = 15 \quad \text{for } y = 0 \\
u & = 20 \quad \text{for } x = 0, x = 8
\end{align*}
\]  

(17)

Following the same technique from the previous section, we arrive at the following system of linear equations:

\[
\sum_{j=1}^{N} [K_{ij}u_j - F_i + \Sigma^e_i] = 0 \quad i = 1, 2, \ldots, N
\]  

(18)

where

\[
K_{ij} = \sum_{e=1}^{N} k^e_{ij} = \sum_{e=1}^{N} \int_{\Omega_e} \left[ k \left( \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y} \frac{\partial \phi_j}{\partial y} \right) \right] dx dy
\]

(19)

and

\[
F_i = \sum_{e=1}^{N} F^e_i = \sum_{e=1}^{N} \int_{\Omega_e} f \phi_i dx dy
\]

(20)

for every element \( e \). For our model problem, the load vector is the zero vector. The contributions of the boundary conditions are imposed through \( \Sigma^e_i = \int_{\partial \Omega_e} \sigma_n \phi_i \ ds \) where \( \sigma_n = -k(x) \nabla u \) and \( k(x) \) is the material modulus.

2.1 Code Implementation

Following the general technique used in the one-dimensional program, we realized that we had to make some minor modifications. First, we need to use a black box called EasyMesh to create a finite element mesh. EasyMesh keeps track of the position of nodal points, elements, and different boundary conditions. Our previous code
was not designed to solve any PDE with boundary conditions. But we did write a specific program to solve the two-dimensional model problem with given boundary conditions. An immediate consequence of our choice of local shape functions is that we no longer needed a numerical integration routine since we were integrating constant functions over triangular domains created using EasyMesh.

As before, we initially programmed the code that computes the global matrix $K_{ij}$ as the sum of the local stiffness matrices. In the same fashion as the one-dimensional case, we programmed the computation of the local Load Vectors of the problem to get the global Load Vector by summing the local Load Vectors. A small difference to the two-dimensional model problem was the existence of $\Sigma_i$ vector. This vector is computed with the same method as the Load Vector: by taking $\sum_{e=1}^{N} \Sigma^e_i$. We used the black box from before to solve (18).

When writing the code, there were many choices to be made. Let’s discuss these two different methods. The CGSolver uses specific data structure to save the matrix and vector data. As a result, before calculating the coefficient vectors, it becomes necessary to save the Stiffness matrix and the Load Vector in the CGSolver structure. Here, we have to choose between the following: 1) perform all the calculations in the native C++ matrix structure and save the results on the CGSolver structure OR 2) perform all the calculations using the CGSolver (which uses less memory than the previous choice but runs much slower because the computer needs to do excessive routine calling during computation).

Using a mesh of about 1180 triangular elements, the program finished computing the approximate solution in about one minute and thirty seconds. Since our future goals consisted of implementing the code with time dependency, we realized that our code needed optimization. This process was simple to implement once we realized that it was not necessary to create a local matrix over each element since this would require memory allocation for $(N - 1) [k^e_{ij}]_{N \times N}$. Therefore, it was natural to create the global matrix first. For each $K_{ij}$ entry, the $i, j$ indices also stand for the location index of a nodal point. Hence, if some element $e$ contains both $i$ and $j$ nodal points, then that element contributes to the entry of the global stiffness matrix using (19). After implementing this algorithm, our program finished its task in approximately twelve seconds.

3 Heat Conduction Equation

Let us turn our attention to the following:
\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{k} \frac{\partial u}{\partial t},
\]

(21)

where \(u\) is now time dependent.

The variational statement for this problem takes the form:

\[
\sum \int \int_{\Omega_e} \left( - \left( \frac{\partial^2 u_e^h}{\partial x^2} + \frac{\partial^2 u_e^h}{\partial y^2} \right) + \frac{1}{k} \frac{\partial u_e^h}{\partial t} \right) v_h^e \, dx \, dy = 0
\]

(22)

where

\[
u_e^h(x,y,t) = \sum_{i=1}^{e} u_i(t) \psi_i^e(x,y),
\]

(23)

and \(v_h^e\) is a test function. Notice that in the approximate solution, the time dependency appears in the value of the functions at the nodes, not on the basis functions.

As before, after manipulating the variational statement, we arrive at the following system of linear equations:

\[
M_{ij} \ddot{u}(t) + K_{ij} u(t) = F_i
\]

(24)

where \(M_{ij}\) denotes the Mass Matrix given locally by:

\[
M_{ij}^e = \int \int_{\Omega_e} \psi_i^e(x,y) \psi_j^e(x,y) \, dx \, dy.
\]

(25)

We solve this system by applying Euler’s Method of Approximation:

\[
M \left( \frac{u(t_{i+1}) - u(t_i)}{\Delta t} \right) + K u(t_{i+1}) = F
\]

(26)

Manipulating (26), we arrive at the following:

\[
(M + \Delta t K) u(t_{i+1}) = \Delta t F + Mu(t_i),
\]

(27)

which requires solving a linear system on every point in the discrete time domain.

Due to time constraint, we were not able to work on the time dependent fluid flow simulation, but it will be easy to implement the code in the future.

7
4 Potential Flow

4.1 The Navier-Stokes Equation

Our goal throughout this project has been to approximate solutions (using the Finite Element Method) to the Navier-Stokes equations for incompressible fluid flow:

\[
\begin{aligned}
\frac{D(\rho u)}{Dt} &= -\nabla p + \nabla \cdot [\mu (\nabla \vec{u} + \nabla \vec{u}^T)] + \rho \vec{g} \\
\nabla \cdot \vec{u} &= 0,
\end{aligned}
\]  

(28)

where \( \vec{u} = (u, v, w) \), and the variables are \( \vec{u} \) and \( p \), velocity and scalar pressure respectively. The symbol \( \frac{D(\rho u)}{Dt} \) represents the total derivative:

\[
\frac{D\phi}{Dt} = \frac{\partial \phi}{\partial t} + \vec{u} \cdot \nabla \phi.
\]  

(29)

Navier-Stokes equations are differential equations which observe the rate of change in velocity and pressure with respect to the position.

Assuming that \( \nabla \times \vec{u} = 0 \) (no vorticity) and the viscous term \( \nabla \cdot [\mu (\nabla \vec{u} + \nabla \vec{u}^T)] \) vanishes, we get to the following set of partial differential equations for the 2-dimensional fluid flow:

\[
\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0, \tag{30}
\]

\[
u = \frac{\partial \phi}{\partial x}, \quad v = \frac{\partial \phi}{\partial y}, \tag{31}
\]

\[
\frac{p}{\rho} + \frac{\| \vec{u} \|}{2} = \text{constant}, \tag{32}
\]

where \( \phi \) is the potential function of the velocity, \( u \) and \( v \) are the components of the velocity, and (32) is Bernoulli’s Equation that relates pressure to velocity.

4.2 Example 1

Consider equation (30) on the \([0, 1] \times [0, 1]\) square subject to the following Dirichlet boundary conditions:

\[
\begin{aligned}
\phi(0, y) &= 0, \forall y \in [0, 1], \\
\phi(x, 0) &= 0, \forall x \in [0, 1], \\
\phi(1, y) &= 0, \forall y \in [0, 1], \\
\phi(x, 1) &= \sin(\pi x), \forall x \in [0, 1],
\end{aligned}
\]  

(33)
This PDE can be numerically solved with the Finite Element Method just as the 2-dimensional model problem. In fact, it is simpler because it does not involve Neumann boundary conditions; it only involves Dirichlet boundary conditions.
Pictured below is the resulting surface for the potential function.

The figure is exactly what we would expect; for \( x \in [0, 1] \) we achieved the sine curve, and zero along the other boundaries. The triangular planes can be seen from this figure.

In order to solve (31), we rewrite the equations in their variational form like we did previously. This results in the following two linear systems of equations (each system for each component of the velocity):

\[
Mu = D^{(u)} \phi \quad Mv = D^{(v)} \phi, \tag{34}
\]

where \( M \) is the Mass Matrix obtained in section 3 and \( D^{(u)} \) and \( D^{(v)} \) can be locally
expressed by
\[ d_{ij}^{(u)} = \frac{\partial \Psi^e_j}{\partial x} \int_{\Omega_e} \Psi^e_j \, dxdy, \quad d_{ij}^{(v)} = \frac{\partial \Psi^e_j}{\partial y} \int_{\Omega_e} \Psi^e_j \, dxdy, \] (35)
for every element.
See the picture below for the velocity field at each nodal point of the mesh.

The last step was to calculate the pressure using Bernoulli’s Equation. For simplicity, we set \( \rho = 1 \) and \( \text{constant} = 0 \). Now, calculating the pressure becomes straightforward. The contours of the pressure can be viewed in the following figure:
4.3 Example 2

The next example simulates the fluid flow on a domain with a circular hole inside. For this case, there will be both Neumann and Dirichlet conditions imposed on the boundaries. The picture below shows the domain of the problem, its triangular discretization and labels to the different boundary conditions.
The conditions are:

\[ \phi = 0 \text{ at } 2; \]
\[ \frac{\partial \phi}{\partial n} = 0 \text{ at } 3; \]
\[ \frac{\partial \phi}{\partial n} = -1 \text{ at } 4. \]

We use the same techniques as in the 2-D model problem to find a FEM approximation for \( \phi \).

The velocity and pressure are calculated as in Example 1. Here are the results:
4.4 Example 3

Consider the following domain:

The boundary conditions are:

\[ \text{at } 2 : \phi = 0; \]
\[ \text{at } 3 : \frac{\partial \phi}{\partial n} = 0; \]
\[ \text{at } 4 : \phi = -50. \]

The next two graphs show the velocity and pressure:
4.5 Code Implementation and Difficulties

Since the Navier-Stokes equations are subject to some hypothesis, we only needed to solve the flow equation, which is exactly the 2-dimensional model problem. To assemble the mass matrix, we used the same algorithm used in calculating the stiffness matrix.

For assembling the velocity field, we solved two linear systems involving the D matrix from the previous section, which was computed in the same way as the mass matrix. The pressure was calculated using (32). Finally, the potential function and the 2-D flux were saved using VTK (Visual Tool Kit).

During the initial runs of the program on examples 2 and 3, the results were wrong. The behavior of the fluid was not physically possible due to the imposed Neumann conditions. In example 2, the flow would enter the hole and in example 3,
the flow would escape through the walls.

After investigating on our thousand line code, we noticed that a little change of sign on the entries of the D-matrix was causing this strange behavior. Once we fixed that, we got the results presented on previous sections, with satisfactory physical behavior.

Acknowledgments The FEM team would like to thank CNPq grant, NSF grant OISE-0526008, USP, IRES, the advisors, Gustavo Nonato and Fabricio S. de Sousa, LCAD, SME, SMA, Maria Helena Noronha and the CSUN Department of Mathematics, Ali Tahzibi and ICMC-USP.

References


