Final Project:
Blasius Boundary Layer and Poisson's Equation

Ryan J. Krattiger*
Missouri University of Science and Technology, Rolla, MO, 65401, USA

*Undergraduate Senior, Department of Mechanical and Aerospace Engineering.

Contents

Listings

Nomenclature

f	Function
$\bar{x}$	Variable value vector
ν	Dynamic viscosity
U	Velocity magnitude
h	Step size
x	x-position
δ	Boundary Layer thickness
$\bar{r}$	residual vector
Subscript	
i	Row index
j	Column index
e	Boundary layer edge

I. Models

I.A. Poisson's Equation

The poisson eqaution is used in many diciplines as it is a common mathematical model found in physical systems. Because of the relative complexity of these physical systems, a simplified and solvable example was used simple to demonstrate the application of two different solution methods.

The first thing to consider is the general form of Poisson's equation as seen in Eq 9. It is clear that this is a second order, linear, partial differential equation in three dimensions, x, y. The forcing function, f(x,y), can be anything depending on the problem and is one of five drivers for the solution.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y) \tag{1}$$

Being a second order PDE, it is required to have two boundary conditions for each independent variable. In this case that would be x and y, resulting in a required four boundary conditions.

For the problem to be solved here, the forcing function can be seen in Eq 10 and the boundary conditions are writen in Eq 11 and q 12.

$$f(x,y) = -2(x^2+y^2) \quad \text{for} \quad (0 \le x \le 1, 0 \le y \le 1) \tag{2}$$

$$u(x,0) = 1 - x^2 \quad , \quad u(x,1) = 2(1-x^2) \quad \text{for} \quad 0 \le x \le 1 \tag{3}$$

$$u(0,y) = 1 + y^2 \quad , \quad u(1,y) = 0 \quad \text{for} \quad 0 \le y \le 1 \tag{4}$$

It can be shown that the exact solution for these constraints is as seen in Eq 13.

$$u(x,y) = (1-x^2)(1+y^2) \tag{5}$$

Reguardless of the forcing function and boundary conditions, the set up of the problem is done so generically, independant of any set of inputs.

This first step is to descretize the original PDE as seen in Eq 9 into the form seen in Eq 14. It can be seen that the second order, central difference scheme was used.

Let $u(x_i, y_j) = u_{i,j}$, where $i \in [0, N]$, $j \in [0, M]$, and $h = \frac{x_N - x_0}{N}$ and $k = \frac{y_M - y_0}{M}$

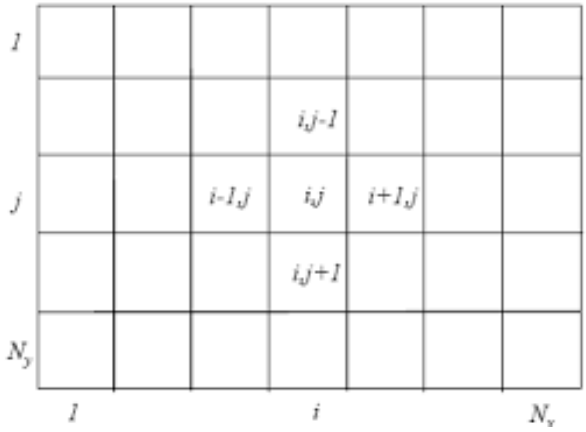
$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} = f(x_i, y_j) \tag{6}$$

Further simplifying this equation and letting $\frac{h^2}{k^2} = \lambda$ gives a general simplified form for the numerical scheme (Eq 15).

$$-2(\lambda+1)u_{i,j} + \lambda u_{i-1,j} + \lambda u_{i+1,j} + u_{i,j-1} + u_{i,j+1} = f(x_i, y_j) \tag{7}$$

After some carefull consideration it can be shown that the above eqation can be applied to the grid structure in figure II.B. It can also be seen that at the boundaries, where the values are known from boundary conditions, can be looked at as constants.

Figure 1. Grid with points used in central difference discretized Poisson



In order to put the now discretized equation into a solvable form it is translated to a system of linear equations. After some work a matrix-vector pair can be found and the system and be written in the form of $A\bar{x} = \bar{b}$

where A is an MxM block diagonal matrix

$$\begin{bmatrix} T & I & 0 & \dots & 0 \\ I & T & I & & \vdots \\ 0 & & \ddots & & 0 \\ \vdots & & I & T & I \\ 0 & \dots & 0 & I & T \end{bmatrix}$$

and T is a NxN tridiagonal matrix

$$\begin{bmatrix} -2(\lambda+1) & \lambda & 0 & \dots & 0 \\ \lambda & -2(\lambda+1) & \lambda & & \vdots \\ 0 & & \ddots & & 0 \\ \vdots & & \lambda & -2(\lambda+1) & \lambda \\ 0 & \dots & 0 & \lambda & -2(\lambda+1) \end{bmatrix}$$

and I is an NxN identity matrix.

In total, A is (NxM)x(NxM) which is quite large. Luckily, there is sparse matrix support in most linear algebra/numerical libraries used in computing to handle this.

The $\bar{b}$ vector is composed of two parts. This first is made up of the boundary conditions ($\bar{c}$), and the second is the forcing vector ($\bar{d}$). The resulting $\bar{b}$ is the denoted as $\bar{b} = \bar{c} + \bar{d}$.

$\bar{c}$ can be defined as

$$\begin{bmatrix} -(u_{1,0}+\lambda u_{0,1}) \\ -u_{2,0} \\ \vdots \\ -(u_{N-1,0}+\lambda u_{N,1}) \\ \lambda u_{0,2} \\ 0 \\ \vdots \\ 0 \\ \lambda u_{N,2} \\ \vdots \\ -(u_{1,M}+\lambda u_{0,M-1}) \\ -u_{2,M-1} \\ \vdots \\ -(u_{N-1,M}+\lambda u_{N,M-1}) \end{bmatrix}$$

$\bar{d}$ can be defined as

$$\begin{bmatrix} f(x_1,y_1) \\ f(x_1,y_2) \\ \vdots \\ f(x_{N-1},y_{M-1}) \end{bmatrix}$$

And finally $\bar{x}$, the solution vector is the set of unknown nodes on the grid. The (i,j) coordinate of the grid is translated using a linear relation to give a vectorized form of the matrix such that

$$\begin{bmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ \vdots \\ u_{N-1,M-1} \end{bmatrix}$$

In the final form of $A\bar{x} = (\bar{c}+\bar{d})$ the Poisson equation is in a form that is solvable. The two methods that will be shown here are the Liebmann Method (also known as the Gauss-Seidel Method) and with a Steepest Decent optimization scheme.

I.A.1. Liebmann Method

As an iterative solving technique, Liebmann Method, also known as the Gauss-Seidel method, is best applied to sparse structured matrices rather than dense matrices. The reason for this is due to the need to perform $O(N^2)$ operations, or one operation for each non zero element of the matrix, to compute the values of $\bar{x}$ in each iteration. If there are many non-zero values, there are more operations required for each iteration. Since the matrix proposed above is indeed sparse, this method is more desirable than a traditional Gauss-Eliminiation technique. Selecting an initial guess, say the $\bar{b}$ vector from above, it is possible to now begin iterations.

For iteration k+1, the general form for the x_i element is Eq 16 for $i \in 0...N$ and N is the size of the solution vector.

$$x_i^{k+1} = \frac{b_i - \sum_{j=0}^{i-1} x_j^{k+1} a_{i,j} - \sum_{j=i+1}^{N} x_j^k a_{i,j}}{a_{i,i}} \tag{8}$$

In implementation the solution vector would be a single variable, so as x_i was updated, the update would be seen when solving for x_{i+1}. With this in mind, the above Eq 16 can be changed into a simpler form seen in Eq 17, recalling that the projection of two vectors (or the dot product) is $\bar{a} \cdot \bar{b} = \sum_{i=0}^{N} a_i b_i$. Also let $\bar{a}_i$ be the vector created by the ith row of the matrix A.

$$x_i = \frac{b_i - \bar{a}_i \cdot \bar{x} + a_{i,i} x_i}{a_{i,i}} \tag{9}$$

By iterating of Eq 17 until convergence of the residual vector $||r_k||_2 \leq tolerance$. The reason to rewrite the solver in this way is to give way to allow the underlying operation of the dot product be optimized for sparse vector types. In this sense, for each product of the A matrix row and the current solution vector, a minimum number of operations will be performed. This also allows for the solver to work for multiple types of matrix problems and still obtain optimal performance without having to write a special solver for every sparse or dense matirx used. The same idea will be used again in the the next solver.

I.A.2. Steepest Decent

Another iterative solver can be used, however the Steepest Decent method looks at this problem as an optimization problem rather than a system of linear equations solving problem.

In general an optimization problem for a function F($\bar{x}$) is as follows:

Determine a search direction $\bar{s}_k$

Solve for the optimal α_k^* to find to minimum in the search direction use the knowns $\bar{x}_k$ and the search direction $\bar{s}_k$ and then use 1D minimization on F($x_{k+!}(\alpha_k^*)$). The below is an example of using the derivative of F to find the minimum.

$$F'(\bar{x}_{k+1}) = 0 \quad \text{where} \quad \bar{x}_{k+1} = \bar{x}_k + \alpha_k^* \bar{s}_k$$

Finally update $\bar{x}$

$$\bar{x}_{k+1} = \bar{x}_k + \alpha_k^* \bar{s}_k$$

and check if the solution meets convergence criteria using the norm of the residual vector ($\bar{r}_k = A\bar{x}_k - \bar{b}$).

$$||r_k||_2 \leq tolerance$$

The current problem to be solved is $Ax - b = \bar{0}$; however, a linear system is not a form that is particularly well suited for optimization since the zero vector is not the minimum. By integrating the system the resulting equation is $\frac{1}{2} x^T A x - x^T b = 0$. This form of the equation does have a minimum as it is a quadratic system of eqations.

For the Steepest Decent method, the search direction is determined by the local direction of steepest decent, or the negative gradient. The formula for this should look familiar as Eq 18, which is the residual vector.

$$\bar{s}_k = \bar{r}_k = \bar{b} - A\bar{x}_k \tag{10}$$

The α_k can be found from the above method finding the root of the derivative as

$$\alpha_k = \frac{s_k \cdot s_k}{s_k \cdot \bar{q}_k}$$

where $\bar{q}_k = A s_k$.

With the search direction (s_k) and optimal α_k^* it is a simple matter to update the solution vector. Again, this will iterator until convergence is found.

There are two additional methods that are linked to Steepest decent. The first is Conjugant Gradient (CGM) and the second is Preconditioned Conjugant Gradient (PCGM). These methods both provide much faster convergence on average, but require more computations per step. More informatoin about these, and a brief study on how they improve the gradient based methods as applied to the solution of 2D Poisson's equation can be found in Holmes et. al.[1]

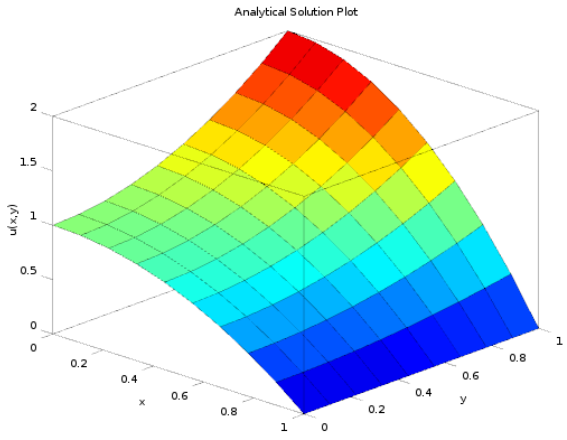
II. Results

The results presented here are simply to show the results and relevent error or convergence of methods used.

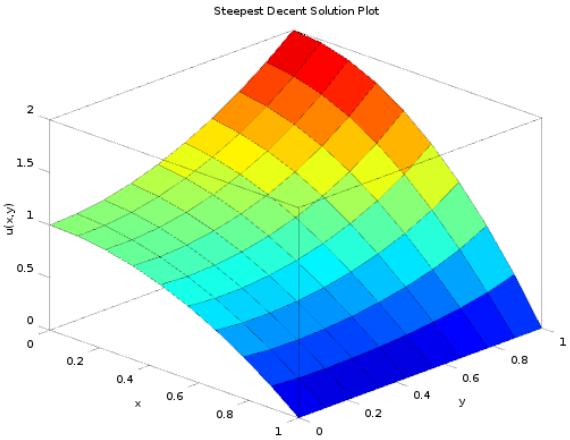
II.A. Poisson's Equation

The figures 3 show the solution plots of each respective method. The plot of the norm of the residual as a function of iteration for each method is presented in figure 4. It can be seen that the steepest decent has a much slower convergence rate compared to the Liebmann method, taking roughly twice as many iterations to converge. The results from comparing the solution method for a 20x20 and 30x30 grid are not included here, but similar conclusions can be drawn.

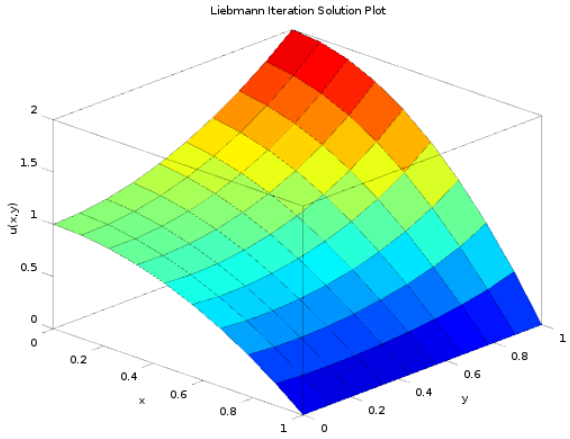
Figure 2. Results from solving the Blasius Boundary Layer equation



(a) Exact Solution



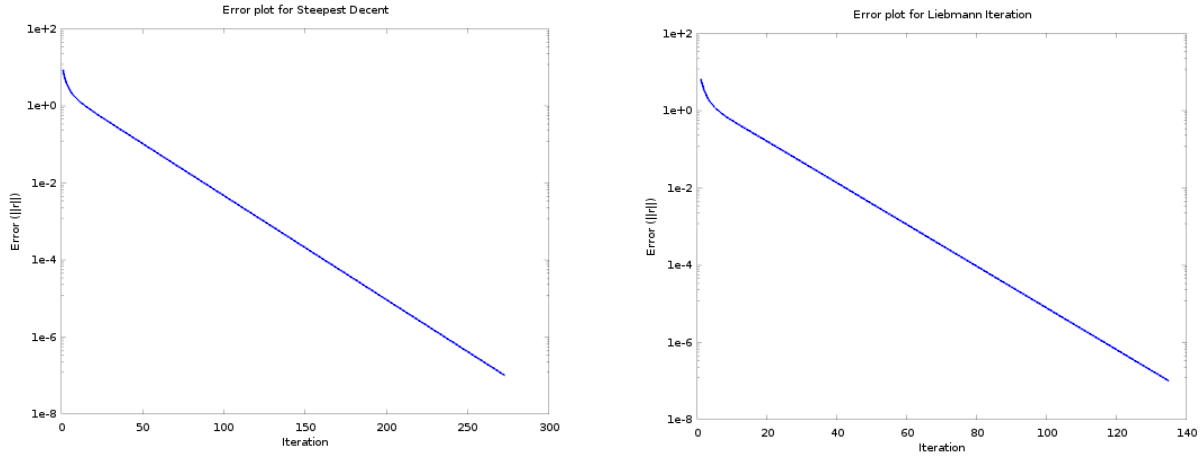
(b) Steepest Decent Solution



(c) Liebmann Solution

Notice that for both methods the number of iterations required is greater than N. It may be tempting to say that this makes the cost of each of these methods roughly equal to that of a Gauss-Elimination as each step appears to be around $O(N^2)$; however, that would be incorrect. Because of the sparse nature of the matrix a good numerical library will leverage the fact that most values are zero and only perform operations on the non-zero values in the matrix and vector. In Matlab, this may not always be the case. In the numerical library that was suppose to be used to solve this problem, and will be shortly after this paper is submitted, the described advantages are leveraged. So the actual complexity is only O(N) operations rather than $O(N^2)$, which puts both of these methods ahead with a total complexity of roughly $O(N^2)$.

Figure 3. Results from solving the Blasius Boundary Layer equation



(a) Steepest Decent error vs iteration

(b) Liebmann error vs iteration

III. Conclusion

References

[1]Holmes, M. H., *Introduction to numerical mehtods in differential equations*, New York, London: Springer, 2007.

[2]Chapra, S. C., and Canale, R. P., *Numerical Methods for Engineers*, McGraw-Hill, 2015.

[3]Numerical methods, – *CFD-Wiki, the free CFD reference* Available: `http://www.cfd-online.com/wiki/numerical_methods`.

[4]Barton, J. J., and Nackman, L. R., *Scientific and engineering C: an introduction with advanced techniques and examples*, Reading, MA: Addison-Wesley, 1994.