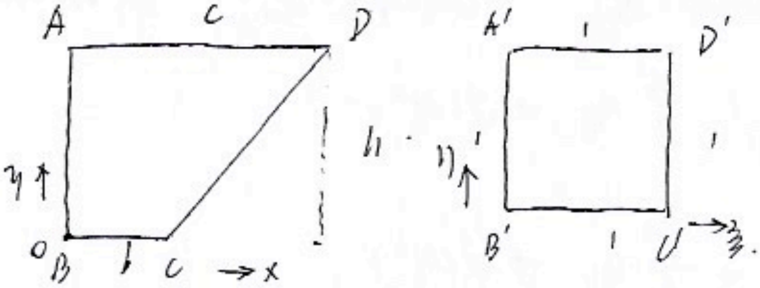
# AME 535a Project 1 Report

1) From $\Omega$ to $\hat{\Omega}$, we can transform the domain by proportion.

for vertical axis coordinate. $\frac{y}{AB} = \frac{\eta}{A'B'} \Rightarrow \frac{y}{h} = \frac{\eta}{1} \Rightarrow y = h\eta$.

for horizontal axis coordinate. $\frac{x}{x_{CD}} = \frac{\xi}{B'C'} \Rightarrow x = \left(\frac{c-b}{h} y + b\right)\xi$




line CD: $x = \frac{c-b}{h}\eta + b$.

Since $y = h\eta$ $\quad x = [(c-b)\eta + b]\xi$.

So mapping T: $\begin{cases} y = h\eta \\ x = [(c-b)\eta + b]\xi \end{cases}$

equation $-\nabla^2 u = 1 \Rightarrow -(u_{xx} + u_{yy}) = 1$ becomes.

$$-\frac{1}{J^2}(a u_{\xi\xi} - 2b u_{\xi\eta} + c u_{\eta\eta} + d u_\eta + e u_\xi) = 1$$

where $a = x_\eta^2 + y_\eta^2$ $\quad b = x_\xi x_\eta + y_\xi y_\eta$ $\quad c = x_\xi^2 + y_\xi^2$ $\quad d = \frac{y_\xi \alpha - x_\xi \beta}{J}$ $\quad e = \frac{x_\eta \beta - y_\eta \alpha}{J}$

$\alpha = a x_{\xi\xi} - 2b x_{\xi\eta} + c x_{\eta\eta}$ $\quad \beta = a y_{\xi\xi} - 2b y_{\xi\eta} + c y_{\eta\eta}$. $\quad J = x_\xi y_\eta - x_\eta y_\xi$

these coefficients can be derived from mapping T.

Boundary Condition:

$u = 0$ on BC and CD stays the same $\Rightarrow$ $u = 0$ on B'C' & C'D'

$\frac{\partial u}{\partial n} = 0$ on AB & AD: ① for AB $n = (1, 0)$

Then $\frac{\partial u}{\partial n} = u_x n^x + u_y n^y = \frac{1}{J}\left[(y_\eta n^x - x_\eta n^y) u_\xi + (-y_\xi n^x + x_\xi n^y) u_\eta\right]$

② Same process for AD $n = (0, 1)$

2) the equation in computational domain is

$$-\frac{1}{J^2}\left(a u_{\xi\xi} - 2b u_{\xi\eta} + c u_{\eta\eta} + d u_\eta + e u_\xi\right) = f = 1$$

for $1^{st}$ order derivatives:

$u_\xi$: Write $u(\xi+\Delta\xi, \eta)$ as $\underbrace{\text{taylor}}_{\text{one variable. } \xi}$ expansion.

and $u(\xi-\Delta\xi,\eta)$ using central difference.

~~$u(\xi,\eta) = u_0 + \Delta\xi\, u_\xi$~~

We denote $u(\xi+\Delta\xi,\eta)$ as $u_{i+1,j}$ $\quad u(\xi,\eta) = u_{i,j}$ $\quad u(\xi-\Delta\xi,\eta) = u_{i-1,j}$ so on ...

then
$$u_{i-1,j} = u_0 - \Delta\xi\, u_\xi + \frac{\Delta\xi^2}{2} u_{\xi\xi} - \frac{\Delta\xi^3}{6} u_{\xi\xi\xi} + O(\Delta\xi^4)$$
$$u_{i,j} = u_0$$
$$u_{i+1,j} = u_0 + \Delta\xi\, u_\xi + \frac{\Delta\xi^2}{2} u_{\xi\xi} + \frac{\Delta\xi^3}{6} u_{\xi\xi\xi} + O(\Delta\xi^4)$$

using method of undeterminal coefficients:

we denote coeffient of $u_{i,j}$ as $\delta_0^1$ where 1 means order of derivative.

0 is the position with recpect to the considered point.

then we have matrix:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & \delta^1_{-1} \\ -1 & 0 & 1 & \delta^1_0 \\ \frac{1}{2} & 0 & \frac{1}{2} & \delta^1_1 \end{array}\right] = \begin{array}{cc} 0 & u_0 \\ 1 & u_\xi \\ 0 & u_{\xi\xi} \end{array} \Rightarrow \begin{pmatrix} -\frac{1}{2} \\ 0 \\ \frac{1}{2} \end{pmatrix}$$

check the order of accuracy:

$2^{nd}$ order derivatives: $-\frac{1}{2}\cdot\frac{1}{2} + 0 + \frac{1}{2}\cdot\frac{1}{2} = 0$

$3^{rd}$ order derivatives: $-\frac{1}{6}\cdot -\frac{1}{2} + \frac{1}{6}\cdot\frac{1}{2} = \frac{1}{6} \neq 0$

so it's $2^{nd}$ order accurate. satisfy requirement.

thus the finite-difference scheme of $u_\xi$ is $\dfrac{-u_{i-1,j} + u_{i+1,j}}{2\Delta\xi}$.

$u_\eta$: Same process as $u_\xi$

and we have $u_\eta = \dfrac{-u_{i,j-1} + u_{i,j+1}}{2\Delta\eta}$

for $2^{nd}$ order derivatives:

$u_{\xi\xi}$: write $u_{i-1,j}$ $u_{i,j}$ $u_{i+1,j}$ in taylor series.

$$u_{i-1,j} = u_0 - \Delta\xi u_\xi + \frac{\Delta\xi^2}{2}u_{\xi\xi} - \frac{\Delta\xi^3}{6}u_{\xi\xi\xi} + \frac{\Delta\xi^4}{24}u_{\xi\xi\xi\xi} + O(\Delta\xi^5)$$

$$u_{i,j} = u_0$$

$$u_{i+1,j} = u_0 + \Delta\xi u_\xi + \frac{\Delta\xi^2}{2}u_{\xi\xi} + \frac{\Delta\xi^3}{6}u_{\xi\xi\xi} + \frac{\Delta\xi^4}{24}u_{\xi\xi\xi\xi} + O(\Delta\xi^5)$$

$$\left.\begin{matrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{matrix}\right| \begin{matrix} \delta'_{-1} \\ \delta'_0 \\ \delta'_1 \end{matrix} = \begin{matrix} 0 \\ 0 \\ 1 \end{matrix} \Rightarrow \begin{vmatrix} 1 \\ -2 \\ 1 \end{vmatrix}$$

check accuracy:

$3^{rd}$ order $-\frac{1}{6}\cdot\frac{1}{2}+\frac{1}{6}\cdot\frac{1}{2}=0$ $\quad$ $4^{th}$ order $\frac{1}{24}\frac{1}{2}+\frac{1}{24}\cdot\frac{1}{2}=\frac{1}{24}\neq 0$.

satisfy $2^{nd}$ accurate.

thus $u_{\xi\xi} = \dfrac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta\xi^2}$.

$u_{\eta\eta}$: same process as $u_{\xi\xi}$ we have $u_{\eta\eta} = \dfrac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta\eta^2}$.

$u_{\xi\eta}$: write $u_{i,j}$, $u_{i-1,j}$ $u_{i+1,j}$ $u_{i,j-1}$ $u_{i,j+1}$ $u_{i+1,j+1}$ $u_{i-1,j-1}$ $u_{i+1,j-1}$ $u_{i-1,j+1}$ in 2 variables taylor series.

$u_{i,j} = u_0$.

$u_{i+1,j} = u_0 + \Delta\xi u_\xi(\xi,\eta) + \frac{\Delta\xi^2}{2}u_{\xi\xi}(\xi,\eta) + \frac{\Delta\xi^3}{6}u_{\xi\xi\xi}(\xi,\eta) + \frac{\Delta\xi^4}{24}u_{\xi\xi\xi\xi}(\xi,\eta) + O(\Delta\xi^5)$

$u_{i-1,j} = 1 - 1 + \frac{1}{2} - \frac{1}{6} + \frac{1}{24}$ (since the term is the same, we just write down the coefficient)

$u_{i,j+1} = u_0 + \Delta\eta u_\eta(\xi,\eta) + \frac{\Delta\eta^2}{2}u_{\eta\eta}(\xi,\eta) + \frac{\Delta\eta^3}{6}u_{\eta\eta\eta}(\xi,\eta) + \frac{\Delta\eta^4}{24}u_{\eta\eta\eta\eta}(\xi,\eta) + O(\Delta\eta^5)$

$u_{i,j-1} = 1 - 1 + \frac{1}{2} - \frac{1}{6} + \frac{1}{24}$ (just coefficient)

$$u_{i+1,j+1} = u_0 + \Delta\xi\, u_\xi + \Delta\eta\, u_\eta + \frac{1}{2}\left(\Delta\xi^2 u_{\xi\xi} + 2\Delta\xi\Delta\eta\, u_{\xi\eta} + \Delta\eta^2 u_{\eta\eta}\right)$$

$$+ \frac{1}{6}\left(\Delta\xi^3 u_{\xi\xi\xi} + 3\Delta\xi^2\Delta\eta\, u_{\xi\xi\eta} + 3\Delta\xi\Delta\eta^2 u_{\xi\eta\eta} + \Delta\eta^3 u_{\eta\eta\eta}\right)$$

$$+ \frac{1}{24}\left(\Delta\xi^4 u_{40} + 4\Delta\xi^3\Delta\eta\, u_{31} + 6\Delta\xi^2\Delta\eta^2 u_{22} + 4\Delta\xi\Delta\eta^3 u_{13} + \Delta\eta^4 u_{04}\right) + O(\Delta\ 5^{th} \text{ order}).$$

$$u_{i-1,j-1} = 1 - 1 - 1 + \frac{1}{2}(1+2+1) + \frac{1}{6}(-1-3-3-1) + \frac{1}{24}(1+4+6+4+1)$$

$$u_{i+1,j-1} = 1 + 1 - 1 + \frac{1}{2}(1-2+1) + \frac{1}{6}(1-3+3-1) + \frac{1}{24}(1-4+6-4+1)$$

$$u_{i-1,j+1} = 1 - 1 + 1 + \frac{1}{2}(1-2+1) + \frac{1}{6}(-1+3-3+1) + \frac{1}{24}(1-4+6-4+1)$$

then we would have equation of coefficients.

$$\left(\begin{array}{ccccccccc|c}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & -1 & 0 & 0 & 1 & -1 & 1 & -1 & 0 \\
0 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 \\
0 & 1 & -1 & 0 & 0 & 1 & -1 & 1 & -1 & 0 \\
0 & 0 & 0 & 1 & -1 & 1 & -1 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 0
\end{array}\right)$$

matrix A B

let x is the vector of coefficients

plug in MATLAB use $x = \mathrm{pinv}(A) * B$

we got $x = (0, 0, 0, 0, 0, \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, -\frac{1}{4})$

and the order of accuracy satisfy.

thus $$u_{\xi\eta} = \frac{u_{i+1,j+1} + u_{i-1,j-1} - u_{i+1,j-1} - u_{i-1,j+1}}{4\Delta\xi\Delta\eta}.$$

Boundary Condition:

for point $B'$ $C'$ $D'$ $u=0$

then $B'$ $u(0,0)=0$ $C'$ $u(1,0)=0$ $D'$ $u(1,1)=0$ in $\xi$-$\eta$ domain.

condition at point $A'$ is the same as $A'B'$

So consider B.C. on $A'B'$ since for $AB$ $n=(1,0)$

in $\xi$-$\eta$ $\frac{\partial u}{\partial n} = \frac{1}{J}\left[(y_\eta n^x - x_\eta n^y) u_\xi + (-y_\xi n^x + x_\xi n^y) u_\eta\right]$

On Boundary point we can't use central difference, so in $\xi$ direction.
we choose forward difference ($2^{nd}$ order)

$$\begin{array}{llllll}
\delta'_0 & u_{i,j} = u_0 & 1 & 1 & 1 & 0 \\
\delta'_1 & u_{i+1,j} = u_0 + \Delta\xi u' + \frac{\Delta\xi^2}{2} u'' + \frac{\Delta\xi^3}{6} u''' & 0 & 1 & 2 & 1 \\
\delta'_2 & u_{i+2,j} = u_0 + 2\Delta\xi u' + \frac{4\Delta\xi^2}{2} u'' + \frac{8\Delta\xi^3}{6} u''' & 0 & \frac{1}{2} & 2 & 0
\end{array} \Rightarrow \begin{pmatrix} -\frac{3}{2} \\ 2 \\ -\frac{1}{2} \end{pmatrix}$$

Thus $u_\xi = \frac{-3u_{i,j} + 4u_{i+1,j} - u_{i+2,j}}{2\Delta\xi}$.

in $\eta$ direction, we choose backward difference ($2^{nd}$ order)

$$\begin{array}{llllll}
\delta'_0 & u_{i,j} = u_0 & 1 & 1 & 1 & 0 \\
\delta'_{-1} & u_{i,j-1} = u_0 - \Delta\eta u' + \frac{\Delta\eta^2}{2} u'' - \frac{\Delta\eta^3}{6} u''' & 0 & -1 & -2 & 1 \\
\delta'_{-2} & u_{i,j-2} = u_0 - 2\Delta\eta u' + \frac{4\Delta\eta^2}{2} u'' - \frac{8\Delta\eta^3}{6} u''' & 0 & \frac{1}{2} & 2 & 0
\end{array} \Rightarrow \begin{pmatrix} \frac{3}{2} \\ -2 \\ \frac{1}{2} \end{pmatrix}$$

thus $u_\eta = \frac{u_{i,j-2} - 4u_{i,j-1} + 3u_{i,j}}{2\Delta\eta}$.

then we can plug in all the terms we need into $\frac{\partial u}{\partial n}$ to get B.C. in $\xi$-$\eta$.

B.C. on $A'D'$ for $AD$ $n=(0,1)$ use same process as $A'B'$

B.C. on $B'C'$ and $C'D'$

Since on $BC$, $CD$ $u=0$ then on $B'C'$, $C'D'$ $u=0$ too.

flow rate $\hat{Q}$:

$$Q = \iint_\phi u\,dx\,dy = \iint_{\hat{\Omega}} u(\xi,\eta)\,|J|\,d\xi\,d\eta \Rightarrow dQ = u(\xi,\eta)\,|J|\,d\xi\,d\eta.$$

In ~~differential~~ discretizied domain, to calculate flow rate in one grid. i choose ~~the~~ to use the central difference to get a mean velocity. of the velocities on 4 corner points ie. $(u_{i,j}+u_{i+1,j}+u_{i-1,j}+u_{i+1,j+1})/4$ and use it as my $\hat{u}(\xi,\eta)$ to calculate flow rate. And same for the Jacobian of transformation. ie $(J_{i,j}+J_{i+1,j}+J_{i,j+1}+J_{i+1,j+1})/4$

So eventually $\hat{Q} = \sum\sum \bar{u}(\xi,\eta)\cdot|\bar{J}|\cdot\Delta\xi\cdot\Delta\eta$.

3) the equation we'll use to solve the problem is:

$$-\frac{1}{J^2}(a u_{\xi\xi} - 2b u_{\xi\eta} + c u_{\eta\eta} + d u_\eta + e u_\xi) = 1$$ Since we know mapping T

$a = x_\eta^2 + y_\eta^2 = (c-b)^2\xi^2 + h^2$ $\quad b = x_\xi x_\eta + y_\xi y_\eta = [(c-b)\eta + b](c-b)\xi.$ $\quad$ where $\xi = (i-1)\Delta\xi$

$c = x_\xi^2 + y_\xi^2 = [(c-b)\eta + b]^2$ $\quad \alpha = a x_{\xi\xi} - 2b x_{\xi\eta} + c x_{\eta\eta} = -2b(c-b)$ $\quad \eta = (j-1)\Delta\eta$

$\beta = 0$ $\quad d = 0$ $\quad e = -\frac{\alpha\cdot h}{J}.$ $\quad J = h[(c-b)\eta + b]$

So in the interior of the domain, after finite differential method, the equation at any given point can be expressed in by linear combination of velocitys of this point and its surrounding point.

$$\cancel{W_{i-1,j-1}\,u_{i-1,j}}\; W_{i-1,j-1}\cdot u_{i-1,j-1} + W_{i,j-1}\cdot u_{i,j-1} + W_{i+1,j-1}\cdot u_{i+1,j-1} + W_{i-1,j}\cdot u_{i-1,j} + W_{i,j}\cdot u_{i,j} + W_{i+1,j}\cdot u_{i+1,j}$$
$$+ W_{i-1,j+1}\cdot u_{i-1,j+1} + W_{i,j+1}\cdot u_{i,j+1} + W_{i+1,j+1}\cdot u_{i+1,j+1} = \hat{f} = 1$$

So now we can plug in the coefficients in the equation and sort out every weight of each component in the discretized equation.

Interior:

$$W_{i+1,j+1} = \cancel{+}\frac{\cancel{+}2b}{4\Delta\xi\,\Delta\eta} \cdot \frac{1}{J^2(i,j)} \quad \text{where } \Delta\xi = \frac{1}{N\_xi-1} \quad \Delta\eta = \frac{1}{N\_eta-1} \quad J = h\left[(c-b)\cdot\Delta\eta(j-1)+b\right]$$

$$\xi = \Delta\xi(i-1) \quad \eta = \Delta\eta(j-1)$$

$$W_{i+1,j} = -\left(\frac{e}{2\Delta\xi} + \frac{a}{\Delta\xi^2}\right)\cdot\frac{1}{J^2(i,j)}$$

$$W_{i+1,j-1} = -\frac{2b}{4\Delta\xi\,\Delta\eta}\cdot\frac{1}{J^2(i,j)} \qquad W_{i,j+1} = -\left(\frac{d}{2\Delta\eta} + \frac{c}{\Delta\eta^2}\right)\cdot\frac{1}{J^2(i,j)}$$

$$W_{i,j} = -\left(\frac{-2a}{\Delta\xi^2} - \frac{2c}{\Delta\eta^2}\right)\cdot\frac{1}{J^2(i,j)} \qquad W_{i,j-1} = -\left(-\frac{d}{2\Delta\eta} + \frac{c}{\Delta\eta^2}\right)\cdot\frac{1}{J^2(i,j)}$$

$$W_{i-1,j+1} = -\frac{2b}{4\Delta\xi\,\Delta\eta}\,\frac{1}{J^2(i,j)} \qquad W_{i-1,j} = -\left(-\frac{e}{2\Delta\xi} + \frac{a}{\Delta\xi^2}\right)\cdot\frac{1}{J^2(i,j)}$$

$$W_{i-1,j-1} = \cancel{-}\,\cancel{+}\frac{2b}{4\Delta\xi\,\Delta\eta}\cdot\frac{1}{J^2(i,j)}$$

Boundary:

Bottom of domain: $W_{i,j} = 1$ for $j = N\_eta$, $i = [2, N\_xi-1]$

Top of domain: for $i = N\_eta$, $j = [2, N\_eta-1]$

$$W_{i,j-2} = \frac{1}{2\Delta\eta\left[(c-b)\eta+b\right]}\cdot\frac{1}{J(i,j)} \qquad W_{i+1,j} = \frac{1}{2\Delta\xi\,(c-b)\,\xi}\,\frac{1}{J(i,j)}$$

$$W_{i,j-1} = -\frac{4}{2\Delta\eta\left[(c-b)\eta+b\right]}\cdot\frac{1}{J(i,j)} \qquad W_{i-1,j} = \frac{-1}{2\Delta\xi\,(c-b)\,\xi}\,\frac{1}{J(i,j)}$$

$$W_{i,j} = \frac{3}{2\Delta\eta\left[(c-b)\eta+b\right]}\cdot\frac{1}{J(i,j)}$$

Left of domain: for $i = 1$, $j = [2, N\_eta-1]$

$$W_{i,j} = -\frac{3h}{2\Delta\xi}\cdot\frac{1}{J(i,j)} \qquad W_{i+1,j} = \frac{4h}{2\Delta\xi}\,\frac{1}{J(i,j)}$$

$$W_{i+2,j} = -\frac{h}{2\Delta\xi}\,\frac{1}{J(i,j)}$$

Right of domain: $W_{i,j} = 1$ for $i = N\_xi$, $j = [2, N\_eta-1]$.

Corner points:

Bottom left: $W_{1,1} = 1$

Bottom right: $W_{N\_xi,1} = 1$

Top left: $W_{1,N.eta} = -\frac{3h}{2\Delta\xi}\frac{1}{J(1,N.eta)}$

$W_{2,N.eta} = \frac{4h}{2\Delta\xi}\frac{1}{J(1,N.eta)}$

$W_{3,N.eta} = -\frac{h}{2\Delta\xi}\frac{1}{J(1,N.eta)}$

Top right: $W_{N.xi,N.eta} = 1$

4). See Ame333a_project1_q4 for the solver of the problem.

the flow scheme calculated by the solver ~~will be~~ is attached to the pdf.

It's the same as the sample solution given in Figure 4.

The calculated flow rate $\hat{Q}$ is 2.7859, which is slightly different from the sample. My speculation is that it's because of different accuracy in the calculating process.

5.) To extend the code to a general Poisson solver for an arbitrary 4-sided solution domain. First, we should set a computation domain with its every parameter. Second, find the mapping to transform from the physical domain to the computational domain. Third, discretized both the LHS and RHS of the Poisson eqt ~~and~~ using finite difference, write every term with its coefficient. RHS should consider input of f. Fourth, ~~construct~~ transfer the B.C. from physical domain to computational domain. Fifth, construct coefficient matrix by stamping and also the RHS vector using the result in 3rd and 4th step.

Sixth, solve the equation by matrix multiplication. Finally, transfer the result in computational domain back to real domain.

6) First, calculate solution and flow rate in each set of $b$s and $N$s. Take when $N = 81$ as reference, ~~so~~ subtract it with the one when $N = 11, 21, 41$, take the absolute value of the result as errors.

for each set of errors vector. apply $\underline{\|e\| = C \cdot \Delta x^{\alpha}}$. take log on
where $C$ is rate and $\alpha$ is convergence order.
both sides of the eqt. then we have $\log \|e\| = \log C + \alpha \log \Delta x$.

for a vector of $\|e\|$ and a vector of its corresponding $\Delta x$. plug into matlab and use the polyfit() function. To get order of converge and convergence rate: We can see that the order of convergence is about 2 and convergence rate are

| log C | Q | L2 | L∞ |
|---|---|---|---|
| $b = 0.5$ | 0.6518 | −0.3159 | ~~−0.3159~~ 0.1093 |
| $b = 0.7$ | 0.7169 | −0.3009 | 0.066. |

See code project1_q6.m for details.

7) Use the for loops in matlab program to vary different sets of parameters and use convhull function. to get Pareto Frontier.

See code project1_q7.m for details.

From the figure we get, if we want achieve the max flow rate, we can choose a point ~~with~~ the which is farest from the I axis on the Pareto Frontier, this point also comes with a relatively large moment of inertia. It makes sense, since big cross section comes with big $I$, and big cross section means bigger flow rate. We can see from the figure, that when $Q$ is relatively small, the points formation are very close and form a approximate linear relation between $Q$ and $I$.

```
function [Solution, FlowRate, I_xx] = ChannelFlow(N_xi, N_eta, ll, bb, hh)
%================================================================================
% ChannelFlow.m
%================================================================================
% The following code is the solution skeleton to the channel flow problem.
% Since this code is intended as a skeleton and should be easy to read,
% we chose not to implement the code using vectorization.
% This will cause slightly lower performance.
%
% Inputs:
% =======
%    N_xi  : Number Of nodes in the xi direction.
%    N_eta : Number of nodes int he eta direction
%    ll    : Channel perimeter
%    bb    : Width of the base of the channel
%    hh    : Height of the channel
%
% Outputs:
% ========
%    Solution : A xi-eta matrix containing the solution
%    Flowrate : The flowrate throught the channel
%    I_xx     : The moment of inertia of the channel
%
% Some additional information for the function:
%
% In this pseudo code, the following conventions are used:
%
%    1) The computational domain xi-eta has values of (0,0) corresponding to the
%       bottom left corner of the physical and computational domains. Correspondingly,
%       the upper-right corner in the xi-eta domain has a (xi, eta) value of (1,1).
%
%    2) We use a node map in the matrix 'Node'. Node(i,j) grabs node reference
%       number. Using the Node matrix, we can easily stamp/stencil the related
%       values into the finite difference matrix. To determine how the Node matrix
%       looks, you can easily type the creation commands in the command prompt.
%
% Originally written by: D.J.Willis
% Modified and distributed by A. Uranga with permission
%
%
%================================================================================
%close all;


%%=== Geometric Parameters =======================================================

cc = bb+((ll-2*bb)^2/4-hh^2)^0.5;



%%=== Grid Details and parameters ================================================

d_xi  = 1./(N_xi-1);
d_eta = 1./(N_eta-1);
NumNodes = N_xi * N_eta;


%%=== Initializations ============================================================

%- Initializing the sparse matrix 'A'
%  Note: It is essential to allocate a sparse A-matrix due to memory restrictions.
A   = spalloc(NumNodes, NumNodes, 9*NumNodes);

%- Initializing the RHS.
RHS = ones(NumNodes,1);


%%=== Note numbering scheme ======================================================
%
% Creation of a node numbering scheme. The node numbering scheme is created here
% in order to simplify the overall solution process. The idea is as follows:
%
% We construct a matrix called Node, which has elements corresponding to the node
```

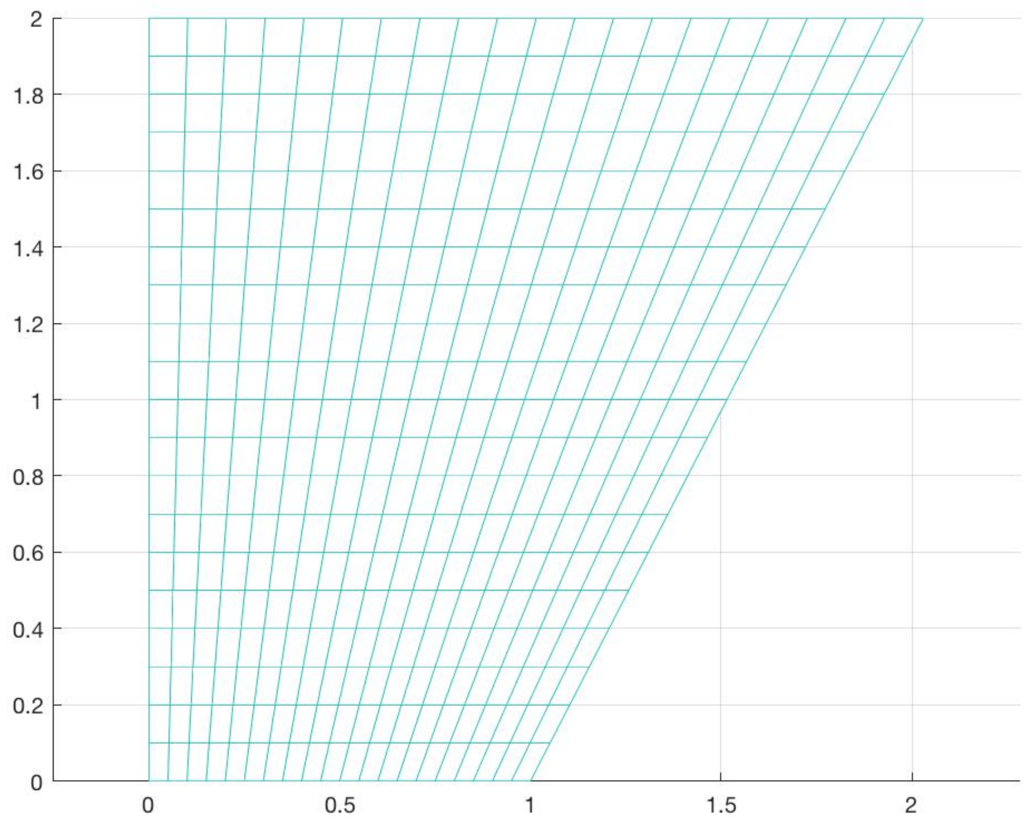
```
% numbers in the grid representation of the solution domain. This allows us to cycle
% through the resulting matrix, grab element (i,j) and easily find the (i +/- 1),
% and (j +/- 1) node numbers.
%
Node = zeros(N_xi, N_eta);
Node(1:NumNodes) = 1:NumNodes;


%%=== Jacobian ==================================================================
'Constructing The Jacobian'

for i = 1:N_xi
   for j = 1:N_eta
      xi(i,j)  = (i-1)*d_xi;
      eta(i,j) = (j-1)*d_eta;
      J(i,j)   = hh*((cc-bb)*eta(i,j)+bb);
   end
end

%%=== "A" Matrix ================================================================
'Constructing the "A" Matrix'

%-------------------------------------------------------------------------------
%------------------------ INNER REGION OF THE DOMAIN ----------------------------
% We begin by considering the Inner region of the domain.
% This is essentially the fill-in for all nodes not touching the boundary.
% The boundary nodes are handled later.

for i = 2:N_xi-1
   for j = 2:N_eta-1
      ANode_i = Node(i,j);            % Setting A-Matrix position for node i,j
        %-----------------------------------------------------------------------------
        %---------------------- The Transformation ---------------------------------
      % The various components of the transformation

        a = (cc-bb)^2*xi(i,j)^2+hh^2;
        b = ((cc-bb)*eta(i,j)+bb)*(cc-bb)*xi(i,j);
        c = ((cc-bb)*eta(i,j)+bb)^2;
        alpha = -2*b*(cc-bb);
        beta  = 0;
        d = 0;
        e = -alpha*hh/J(i,j);

        %-----------------------------------------------------------------------------
        %---------------------- FILLING UP THE A MATRIX ----------------------------
        % The filling of the matrix is done via the stamping of the computational
      % molecule in the appropriate parts of the A-Matrix.


        %-----------------------------------------------------------------------------
        %----------------------  RHS Part Of Computational Molecule  -----------------
        % Here A(p,q) is such that:
      %    p = the current node number on the grid, at which the
        %        differential equation is being approximated
      %    q = the neighboring point to the current node.
      % So, here we are using a stamping stencil based on the ANode_i matrix.
      % It may be worthwhile to display a reduced dimension version of ANode_i
      % to fully grasp what is happening here.

      A(ANode_i, Node(i+1, j+1) ) = -(-2*b/(4*d_xi*d_eta))/J(i,j)^2;
      A(ANode_i, Node(i+1, j  ) ) = -(e/(2*d_xi)+a/d_xi^2)/J(i,j)^2;
      A(ANode_i, Node(i+1, j-1) ) = -(2*b/(4*d_xi*d_eta))/J(i,j)^2;


      %-----------------------------------------------------------------------------
      %--------------------  Middle Part Of Computational Molecule  ---------------

      A(ANode_i, Node(i  , j+1) ) = -(d/(2*d_eta)+c/d_eta^2)/J(i,j)^2;
      A(ANode_i, Node(i  , j  ) ) = -(-2*a/d_xi^2-2*c/d_eta^2)/J(i,j)^2;
      A(ANode_i, Node(i  , j-1) ) = -(-d/(2*d_eta)+c/d_eta^2)/J(i,j)^2;


      %-----------------------------------------------------------------------------
```

```matlab
      %----------------------  LHS  Part Of Computational Molecule  ----------------

      A(ANode_i, Node(i-1, j+1) ) = -(2*b/(4*d_xi*d_eta))/J(i,j)^2;
      A(ANode_i, Node(i-1, j  ) ) = -(-e/(2*d_xi)+a/d_xi^2)/J(i,j)^2;
      A(ANode_i, Node(i-1, j-1) ) = -(-2*b/(4*d_xi*d_eta))/J(i,j)^2;

    end
end

%----------------------------------------------------------------------------------
%------------------------ BOUNDARY CONDITIONS -------------------------------------

%------------------------ Bottom of the domain ------------------------------------
j = 1;
for i = 2:N_xi-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i,j)) = 1;
   RHS(ANode_i) = 0;
end

%------------------------ Top of the domain ---------------------------------------
j = N_eta;
for i = 2:N_xi-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i, j-2) ) = 1/(2*d_eta)*((cc-bb)*eta(i,j)+bb)/J(i,j);
   A(ANode_i, Node(i, j-1) ) = -4/(2*d_eta)*((cc-bb)*eta(i,j)+bb)/J(i,j);
   A(ANode_i, Node(i, j  ) ) = 3/(2*d_eta)*((cc-bb)*eta(i,j)+bb)/J(i,j);
   A(ANode_i, Node(i-1, j) ) = 1/(2*d_xi)*(cc-bb)*xi(i,j)/J(i,j);
   A(ANode_i, Node(i+1, j) ) = -1/(2*d_xi)*(cc-bb)*xi(i,j)/J(i,j);
   RHS(ANode_i) = 0;
   % Fill in the boundary condition for the top of the domain

end

%------------------------ Left side of the domain ---------------------------------
i = 1;
for j = 2:N_eta-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i  , j) ) = -3/(2*d_xi)*hh/J(i,j);
   A(ANode_i, Node(i+1, j) ) = 4/(2*d_xi)*hh/J(i,j);
   A(ANode_i, Node(i+2, j) ) = -1/(2*d_xi)*hh/J(i,j);
   RHS(ANode_i) = 0;
   % Fill in the boundary condition for the left of the domain

end

%------------------------ Right side of the domain --------------------------------
i = N_xi;
for j = 2:N_eta-1
   ANode_i = Node(i,j);
   A(ANode_i, Node(i,j)) = 1;
   RHS(ANode_i) = 0;
   % Fill in the boundary condition for the Right side of the domain

end

%------------------------ DOMAIN CORNERS ------------------------------------------

%------------------------ Bottom left ---------------------------------------------
ANode_i = Node(1,1);
A(ANode_i, Node(1,1)) = 1;
RHS(ANode_i) = 0;
% Fill in the boundary condition for the bottom left corner of the domain


%------------------------ Bottom right --------------------------------------------
ANode_i = Node(N_xi,1);
A(ANode_i, Node(N_xi,1)) = 1;
RHS(ANode_i) = 0;
% Fill in the boundary condition for the bottom right corner of the domain


%------------------------ Top Left ------------------------------------------------
```

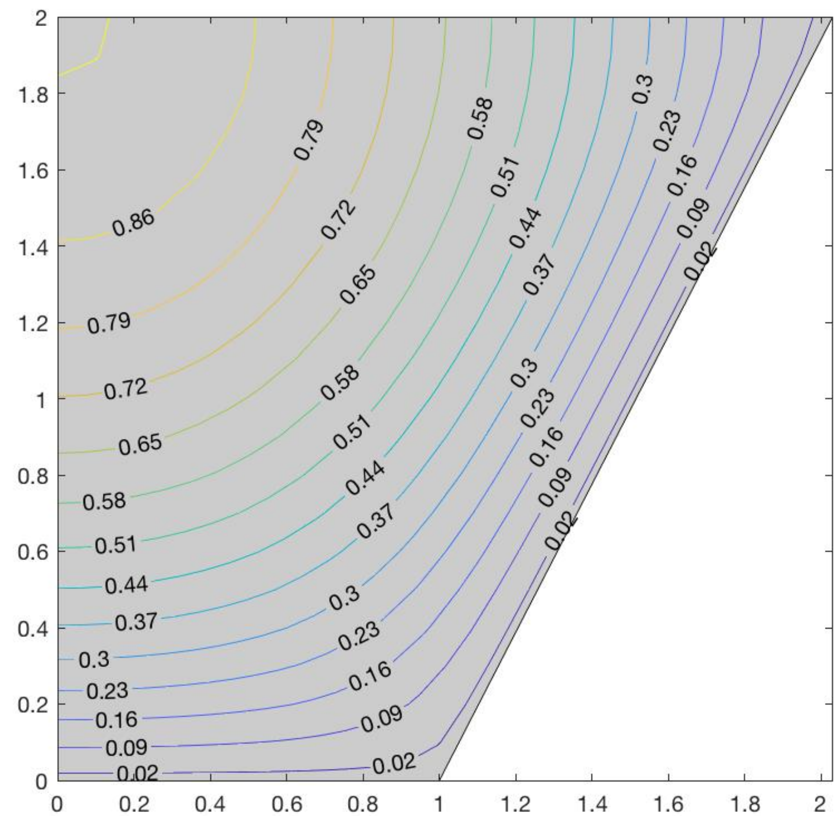
```
ANode_i = Node(1,N_eta);  % Setting A_Matrix position for node i,j
A(ANode_i, Node(1  , N_eta) ) = -3/(2*d_xi)*hh/J(1,N_eta);
A(ANode_i, Node(1+1, N_eta) ) = 4/(2*d_xi)*hh/J(1,N_eta);
A(ANode_i, Node(1+2, N_eta) ) = -1/(2*d_xi)*hh/J(1,N_eta);
RHS(ANode_i) = 0;
% Fill in the boundary condition for the top left corner of the domain


%------------------------ Top right ----------------------------------------------
ANode_i = Node(N_xi,N_eta);
A(ANode_i, Node(N_xi,N_eta)) = 1;
RHS(ANode_i) = 0;
% Fill in the boundary condition for the top right corner of the domain


%%=== Solving the system Ax=b ====================================================
'Solving the system'

Sol = A\RHS;
Solution = reshape(Sol,N_xi,N_eta);


%%=== Post-processing the solution ===============================================
'Post-processing'

%--------------------------------------------------------------------------------
%------------------------ Computing the flow rate & moment of inertia -------------
% With the velocity known, we can compute the flowrate and the moment of inertia.
% As a check of your flow rate integral try computing the area of the channel
% and compare that with the analytical result.

%-- Flow rate
FlowRate = 0;
for i = 1:N_xi-1
    for j = 1:N_eta-1
        FlowRate =
FlowRate+(Solution(i,j)+Solution(i+1,j)+Solution(i,j+1)+Solution(i+1,j+1))/4*...
        (J(i,j)+J(i+1,j)+J(i,j+1)+J(i+1,j+1))/4*d_xi*d_eta;
    end
end
FlowRate = FlowRate*2;
% Fill In Your Computations For the Flow Rate


%-- Moment of inertia
t = 0.05;
I_xx = bb*t^3/6+bb*t*hh^2/2+t*((cc-bb)^2+hh^2)^0.5/6*(hh^2+t^2*(cc-bb)^2/((cc-bb)^2+hh^2));
% Fill In Your Computations For the Moment Of Inertia



%%=== Information for the plots ==================================================
% Fill-in the transformation equations for x(xi,eta) and y(xi,eta)
for i = 1 : N_xi
   for j = 1 : N_eta
      xi  = (i-1)*d_xi;
      eta = (j-1)*d_eta;
      x(i,j)= ((cc-bb)*eta+bb)*xi;
      y(i,j)= eta*hh;
   end
end


%%=== Solution output ============================================================
% Uncomment the following lines to make the plots
%
% %==============================================================================
% %-- Plot the mesh
figure,
mesh(x,y,0*x,0*y);
view([0,0,1]);
axis equal
%
```

```
% %==============================================================================
% %-- Plot the solution
figure,
[ccc,fff]=contour(x,y,abs(Solution),(0.02:0.07:max(max(abs(Solution))))');
clabel(ccc,fff);
hold on
patch([0,bb,cc,0],[0,0,hh,hh],-ones(1,4),0,'facecolor',[0.8,0.8,0.8]);
axis equal
```

# Grid

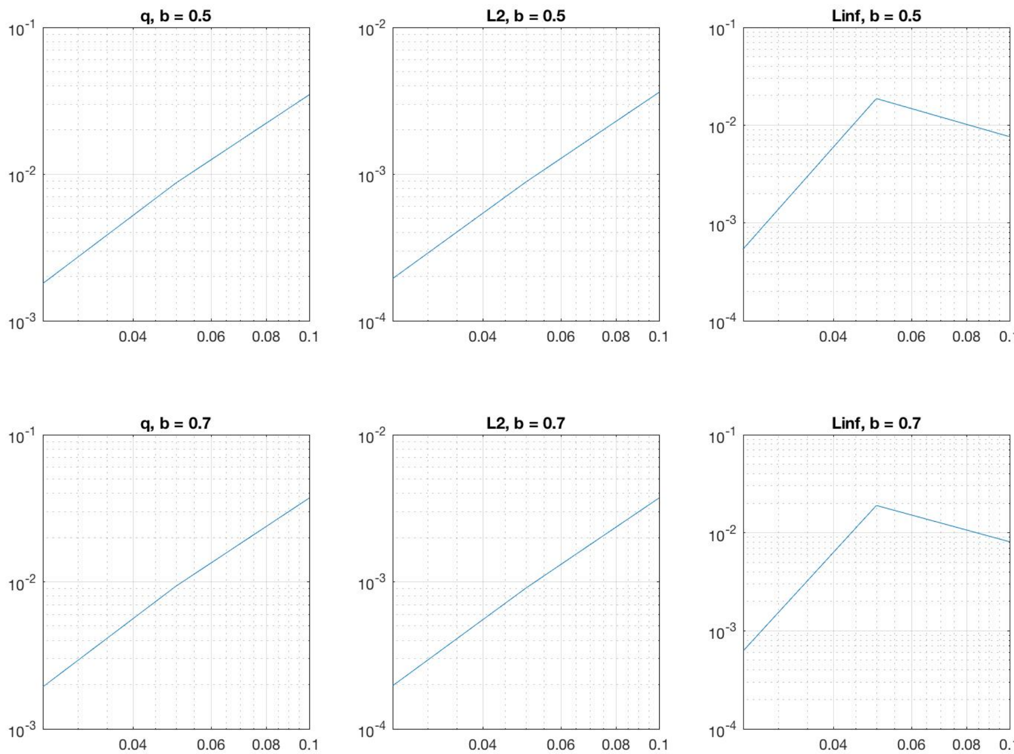


# Flow scheme


0.86
0.79
0.72
0.65
0.58
0.51
0.44
0.37
0.3
0.23
0.16
0.09
0.02
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

```matlab
% case bb = 0.5
[q81_1,sol81_1] = Q_Sol_output(81,81,6.5,0.5,2);
[q41_1,sol41_1] = Q_Sol_output(41,41,6.5,0.5,2);
[q21_1,sol21_1] = Q_Sol_output(21,21,6.5,0.5,2);
[q11_1,sol11_1] = Q_Sol_output(11,11,6.5,0.5,2);

% restrict the solution obtained at the reference mesh to the current coarse mesh
for i = 1:11
    for j = 1:11
        e11_1(i,j) = abs(sol11_1(i,j)-sol81_1(1+(i-1)*8,1+(j-1)*8));
    end
end

for i = 1:21
    for j = 1:21
        e21_1(i,j) = abs(sol21_1(i,j)-sol81_1(1+(i-1)*4,1+(j-1)*4));
    end
end

for i = 1:41
    for j = 1:41
        e41_1(i,j) = abs(sol41_1(i,j)-sol81_1(1+(i-1)*2,1+(j-1)*2));
    end
end

% reshape solution matrix into vector
e11_1 = e11_1(:);
e12_1 = e21_1(:);
e41_1 = e41_1(:);

% calculate the errors of the flowrate
eq_11_1 = abs(q11_1-q81_1);
eq_21_1 = abs(q21_1-q81_1);
eq_41_1 = abs(q41_1-q81_1);

eq1 = [eq_11_1,eq_21_1,eq_41_1];
xeq1 = [1/10,1/20,1/40];
a1_1 = polyfit(log10(xeq1),log10(eq1),1);

% calculate the errors of the L2 norm

eL2_11_1 = 1/10*norm(e11_1,2);
eL2_21_1 = 1/20*norm(e21_1,2);
eL2_41_1 = 1/40*norm(e41_1,2);

eL2_1 = [eL2_11_1,eL2_21_1,eL2_41_1];
xeL2_1 = [1/10,1/20,1/40];
a2_1 = polyfit(log10(xeL2_1),log10(eL2_1),1);

% calculate the errors of the Linf norm
eLinf_11_1 = norm(e11_1,inf);
eLinf_21_1 = norm(e21_1,inf);
eLinf_41_1 = norm(e41_1,inf);

eLinf_1 = [eLinf_11_1,eLinf_21_1,eLinf_41_1];
xeLinf_1 = [1/10,1/20,1/40];
a3_1 = polyfit(log10(xeLinf_1),log10(eLinf_1),1);

% case bb = 0.7
[q81_2,sol81_2] = Q_Sol_output(81,81,6.5,0.7,2);
[q41_2,sol41_2] = Q_Sol_output(41,41,6.5,0.7,2);
[q21_2,sol21_2] = Q_Sol_output(21,21,6.5,0.7,2);
[q11_2,sol11_2] = Q_Sol_output(11,11,6.5,0.7,2);

% restrict the solution obtained at the reference mesh to the current coarse mesh
for i = 1:11
    for j = 1:11
        e11_2(i,j) = abs(sol11_2(i,j)-sol81_2(1+(i-1)*8,1+(j-1)*8));
    end
end

for i = 1:21
    for j = 1:21
```

```
        e21_2(i,j) = abs(sol21_2(i,j)-sol81_2(1+(i-1)*4,1+(j-1)*4));
    end
end

for i = 1:41
    for j = 1:41
        e41_2(i,j) = abs(sol41_2(i,j)-sol81_2(1+(i-1)*2,1+(j-1)*2));
    end
end

% reshape solution matrix into vector
e11_2 = e11_2(:);
e12_2 = e21_2(:);
e41_2 = e41_2(:);

% calculate the errors of the flowrate
eq_11_2 = abs(q11_2-q81_2);
eq_21_2 = abs(q21_2-q81_2);
eq_41_2 = abs(q41_2-q81_2);

eq2 = [eq_11_2,eq_21_2,eq_41_2];
xeq2 = [1/10,1/20,1/40];
a1_2 = polyfit(log10(xeq2),log10(eq2),1);

% calculate the errors of the L2 norm

eL2_11_2 = 1/10*norm(e11_2,2);
eL2_21_2 = 1/20*norm(e21_2,2);
eL2_41_2 = 1/40*norm(e41_2,2);

eL2_2 = [eL2_11_2,eL2_21_2,eL2_41_2];
xeL2_2 = [1/10,1/20,1/40];
a2_2 = polyfit(log10(xeL2_2),log10(eL2_2),1);

% calculate the errors of the Linf norm
eLinf_11_2 = norm(e11_2,inf);
eLinf_21_2 = norm(e21_2,inf);
eLinf_41_2 = norm(e41_2,inf);

eLinf_2 = [eLinf_11_2,eLinf_21_2,eLinf_41_2];
xeLinf_2 = [1/10,1/20,1/40];
a3_2 = polyfit(log10(xeLinf_2),log10(eLinf_2),1);

% plot
subplot(2,3,1)
loglog(xeq1,eq1);
title('q, b = 0.5')
grid on

subplot(2,3,2)
loglog(xeL2_1,eL2_1);
title('L2, b = 0.5')
grid on

subplot(2,3,3)
loglog(xeLinf_1,eLinf_1);
title('Linf, b = 0.5')
grid on

subplot(2,3,4)
loglog(xeq2,eq2);
title('q, b = 0.7')
grid on

subplot(2,3,5)
loglog(xeL2_2,eL2_2);
title('L2, b = 0.7')
grid on

subplot(2,3,6)
loglog(xeLinf_2,eLinf_2);
title('Linf, b = 0.7')
grid on
```


q, b = 0.5
L2, b = 0.5
Linf, b = 0.5
q, b = 0.7
L2, b = 0.7
Linf, b = 0.7
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
0.04
0.06
0.08
0.1

```
i = 0;
for h = 0.1:0.1:2
    for b = 0.1:0.1:2
        if h+b <= 3.25
        i = i+1;
        [Q(i),I(i)] = Q_I_output(21,21,6.5,b,h);
        end
    end
end
k = convhull(Q,I);
plot(Q(k),I(k),Q,I,'.');
```