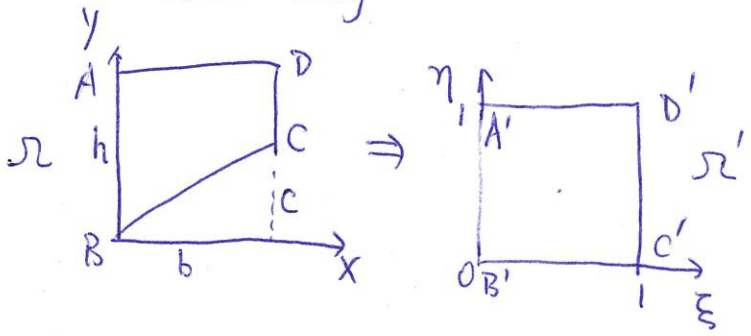
Project 1 Hao Tang

1.



The mapping gives $\begin{cases} x = b\xi \\ y = c\xi + (h - c\xi)\eta \end{cases}$, $\begin{cases} \xi = \frac{1}{b}x \\ \eta = \frac{1}{h - \frac{c}{b}x}(y - \frac{c}{b}x) \end{cases}$

The equation is $-\nabla^2 u = 1$, $\Rightarrow$ $u_{xx} + u_{yy} = -1$

we have $\partial_x = \frac{\partial \xi}{\partial x}\partial_\xi + \frac{\partial \eta}{\partial x}\partial_\eta = \frac{1}{b}\partial_\xi + \frac{-\frac{c}{b}(h - \frac{c}{b}x) + \frac{c}{b}(y - \frac{c}{b}x)}{(h - \frac{c}{b}x)^2}\partial_\eta$

$= \frac{1}{b}\partial_\xi + \frac{c}{(hb - cx)^2}[b(y-h) + 2cx]\partial_\eta$

$\partial_y = \frac{\partial \xi}{\partial y}\partial_\xi + \frac{\partial \eta}{\partial y}\partial_\eta = \frac{1}{h - \frac{c}{b}x}\partial_\eta$

Then $u_{xx} = \frac{1}{b}\partial_\xi^2 + \frac{c}{b(h - c\xi)^2}$

then we have $x_\eta = 0$, $x_\xi = b$, $y_\eta = h - c\xi$, $y_\xi = c(1-\eta)$

$J = x_\xi y_\eta - x_\eta y_\xi = b(h - c\xi)$ the equation gives.

$a = (h - c\xi)^2$, $b = c(h - c\xi)(1-\eta)$, $c = b^2 + c^2(1-\eta)^2$

$x_{\xi\xi} = 0$, $x_{\xi\eta} = 0$, $x_{\eta\eta} = 0$, $y_{\xi\xi} = 0$, $y_{\xi\eta} = -c$, $y_{\eta\eta} = 0$

$\Rightarrow$ $\beta = -2b \cdot (-c) = 2c^2(h - c\xi)(1-\eta)$ $\alpha = 0$. $\Rightarrow$ $d = \frac{x_\eta \beta}{J} = \frac{2bc \cdot 0}{b(h - c\xi)} = 0$

$e = \frac{-x_\xi \beta}{J} = \frac{-2c^2 \cdot b(h - c\xi)(1-\eta)}{b(h - c\xi)} = -2c^2(1-\eta)$

The equation is then

$$-\frac{1}{b^2(h-c\xi)^2}\Big[(h-c\xi)^2 u_{\xi\xi} - 2c(h-c\xi)(1-\eta)\,u_{\xi\eta} + (b^2+c^2(1-\eta)^2)\,u_{\eta\eta} - 2c^2(1-\eta)\,u_\eta\Big] = 1$$

Boundary condition $u|_{\xi=1} = u|_{\eta=0} = 0$

for AB boundary:

$\vec{n} = (1,0)$ $\quad \frac{\partial u}{\partial n} = \frac{1}{J}\left[y_\eta n^x u_\xi - y_\xi n^x u_\eta\right] = \frac{(h-c\xi)u_\xi - c(1-\eta)u_\eta}{b(h-c\xi)}$

$$\Rightarrow \left(\frac{1}{b}u_\xi - \frac{c(1-\eta)}{b(h-c\xi)}u_\eta\right)\Big|_{\xi=0} = 0$$

for AD boundary $\vec{n} = (0,1)$ $\quad \frac{\partial u}{\partial n} = \frac{1}{J}\left[-x_\eta u_\xi + x_\xi u_\eta\right] = \frac{1}{(h-c\xi)}u_\eta$

$$\Rightarrow \quad u_\eta|_{\eta=1} = 0$$

2. Interior of domain:

$$-\frac{1}{b^2}\frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{\Delta\xi^2} + \frac{2c(1-\eta_j)}{b^2(h-c\xi_i)}\cdot\frac{u_{i+1,j+1}+u_{i-1,j-1}-u_{i+1,j-1}-u_{i-1,j+1}}{4\Delta\xi\Delta\eta}$$

$$-\frac{b^2+c^2(1-\eta_j)^2}{b^2(h-c\xi_i)^2}\frac{u_{i,j+1}-2u_{ij}+u_{i,j-1}}{\Delta\eta^2} + \frac{2c^2(1-\eta)}{b^2(h-c\xi_i)^2}\frac{u_{i,j+1}-u_{i,j-1}}{2\Delta\eta} = 1$$

For boundary points: BC: $u_{i,0} = 0$; CD: $u_{N-1,j} = 0$

AD: $\frac{u_{i,N} - u_{i,N-1}}{2\Delta\eta} = 0$; AB: $\frac{1}{b}\frac{u_{j} - u_{-1,j}}{2\Delta\xi} - \frac{c(1-\eta_j)}{bh}\frac{u_{0,j+1} - u_{0,j-1}}{2\Delta\eta} = 0$

Here we extend $i = -1, 0, 1, \cdots, N-1$; $j = 0, 1, \cdots, N-1, N$

Numerically, we have $Q=\int 2u\,dx\,dy = 2\int_0^1\int_0^1 u(\xi,\eta)\,J(\xi,\eta)\,d\xi\,d\eta$

$$=2\int_0^1\int_0^1 u(\xi,\eta)\,b(h-c\xi)\,d\xi\,d\eta$$

$$= 2\Bigg[\sum_{i=1}^{N-1}\sum_{j=0}^{N-2} u_{ij}\,b(h-c\xi_i)\,\Delta\xi\,\Delta\eta + \frac{1}{2}\sum_{j=0}^{N-2} u_{0j}\,b(h-\xi_0)\,\Delta\xi\,\Delta\eta + \frac{1}{2}\sum_{i=0}^{N-1} u_{i,N-1}\,b(h-\xi_i)\,\Delta\xi\,\Delta\eta\Bigg]$$

$$= 2b\,\Delta\xi\,\Delta\eta\left[\sum_{i=1}^{N-1}\sum_{j=0}^{N-2} u_{ij}\left(h-c\frac{i}{N-1}\right) + \frac{h}{2}\sum_{j=0}^{N-2} u_{0j} + \frac{1}{2}\sum_{i=0}^{N-1} u_{i,N-1}\left(h-c\frac{i}{N-1}\right)\right]$$

3. Here we turn the equation into matrix form: The row corresponds to N_{ij} (interier node) is:

$$[0,\cdots,0,A_{i-1,j-1},A_{i-1,j},A_{i-1,j+1},0,\cdots,0,A_{ij-1},A_{ij},A_{ij+1},0,\cdots,0,A_{i+1,j-1},A_{i+1,j},A_{i+1,j+1},0,\cdots 0]$$

where $A_{i-1,j-1} = A_{i+1,j+1} = -A_{i-1,j+1} = -A_{i+1,j-1} = \dfrac{2c(1-\eta_j)}{b^2(h-c\xi_i)}\cdot\dfrac{1}{4\Delta\xi\Delta\eta}$

$$A_{i-1,j} = A_{i+1,j} = -\frac{1}{b^2\Delta\xi^2},\qquad A_{i,j-1} = -\frac{b^2+c^2(1-\eta_j)^2}{b^2(h-c\xi_i)^2}\frac{1}{\Delta\eta^2} - \frac{2c^2(1-\eta)}{b^2(h-c\xi_i)^2}\frac{1}{2\Delta\eta}$$

$$A_{ij} = \frac{2}{b^2\Delta\xi^2} + 2\,\frac{b^2+c^2(1-\eta_j)^2}{b^2(h-c\xi_i)^2}\frac{1}{\Delta\eta^2} \qquad A_{i,j+1} = -\frac{b^2+c^2(1-\eta_j)^2}{b^2(h-c\xi_i)^2}\frac{1}{\Delta\eta^2} + \frac{2c^2}{b^2(h-c\xi_i)^2}\frac{1}{2\Delta\eta}$$

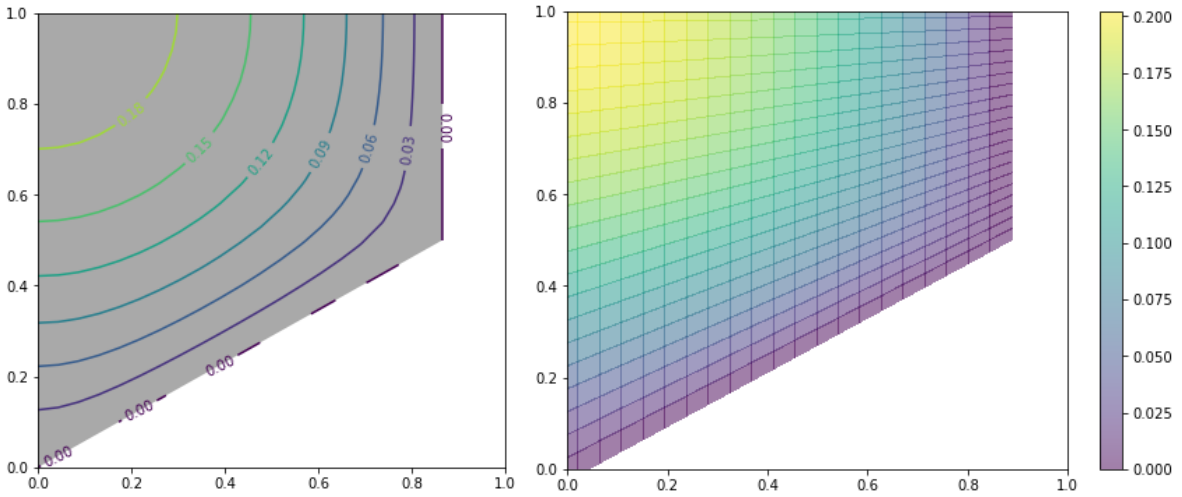
for boundary nodes: BC, N_{i0} : $A_{i0}=1$, all others entries are 0

CD, $N_{N-1,j}$: $A_{N-1,j}=1$, all others entries are 0

AD, $N_{i,N}$: $A_{iN} = -A_{i,N-2} = \dfrac{1}{2\Delta\eta}$

AB, $N_{-1,j}$: $A_{1j} = -A_{-1,j} = \dfrac{1}{2b\Delta\xi}$, $A_{0,j+1} = -A_{0,j-1} = -\dfrac{c(1-\eta_j)}{2bh\Delta\eta}$

4. The code is attached as 1.4.py and the simulation results are as follow:



The derived value of $Q = 0.129$. The derived results are in perfect agreement with the sample solution.

5. To extend the program into a Poisson solver of arbitrary 4-side problem, we can

a. Take arbitrary transformation $(x, y) \rightarrow (\xi, \eta)$ as inputs. The input specifies the Jacobi matrix $J(\xi, \eta)$.

b. Calculate $|J|$ and coefficients in $\boxed{\frac{-1}{J^2}\left(a\, u_{\xi\xi} - 2b\, u_{\xi\eta} + c\, u_{\eta\eta} + d\, u_{\xi} + e\, u_{\eta}\right) = f}$.

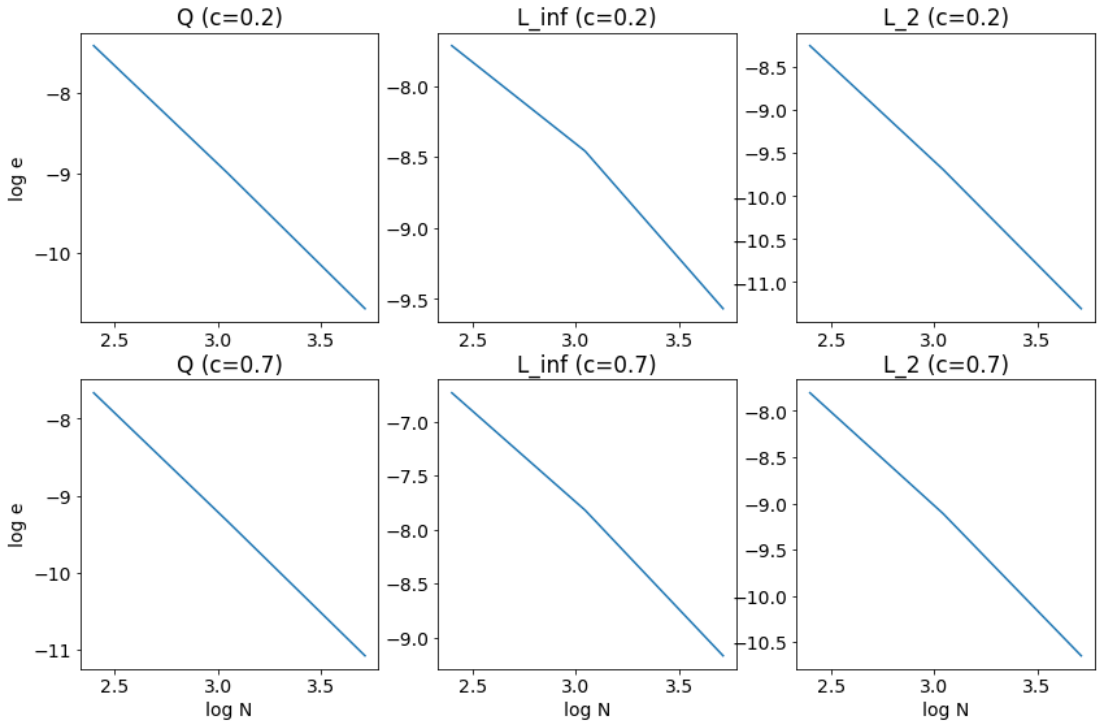
c. Write down finite difference expressions of $u_{\xi\xi}, u_{\eta\eta}, u_{\eta\xi}, u_{\xi}, u_{\eta}$

d. Turns the equations into matrix form and calculate the matrix A and f using the obtained coefficients

e. Solve equation to obtain $u(\xi, \eta)$

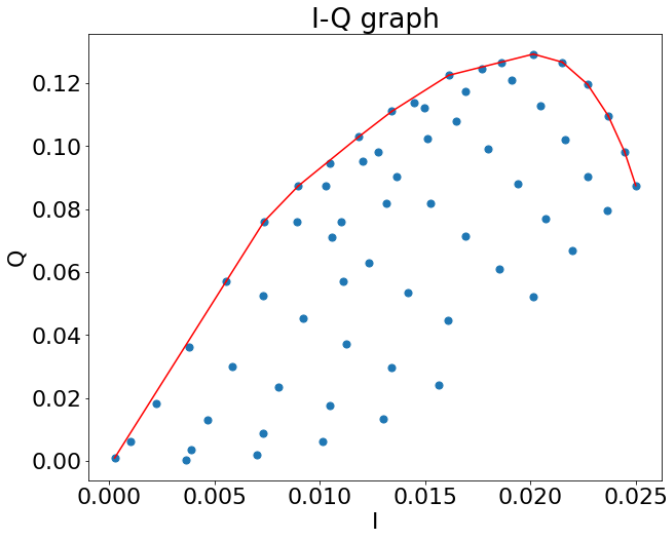
f. Transform back $u(x, y) = u(\xi(x, y), \eta(x, y))$, and calculate $Q = \int d\eta d\xi\, |J| u(\eta, \xi)$

6. The log-log plot of errors Q, L_∞, and L_2 for the two different c values are shown below.



We can see that in all plots, the slopes of the lines are approximately -2, showing the second order convergence. Codes that generate this plot is 1.6.py.

7. Here, we plot the calculated I and Q in the graph below.



The convex Hull of these points are shown as the red lines. The Pareto optimal choices gives maximal flux for given inertia, as shown on the graph. The resulting geometry shows that for $I < 0.02$, we must

increase I in order to increase Q. While within the data range, the value of Q has a maximum. After $I \sim 0.02$, further increasing I will lead to decreasing flux Q.

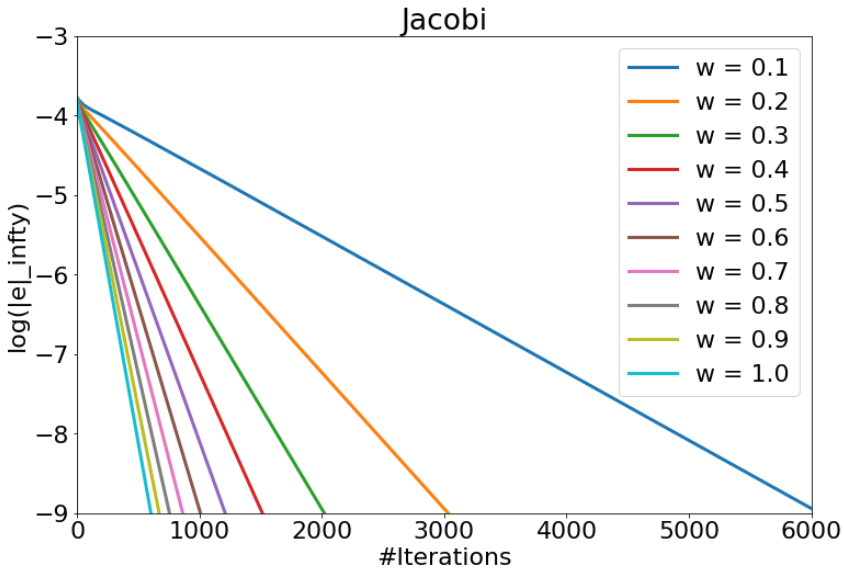
	h	c
1	0.6	0.0
2	0.7	0.0
3	0.9	0.3
4	1.0	0.5
5	1.0	0.8
6	1.0	0.7
7	1.0	0.6
8	0.5	0.0
9	0.1	0.0
10	0.8	0.2
11	1.0	0.9
12	1.0	1.0
13	0.8	0.1

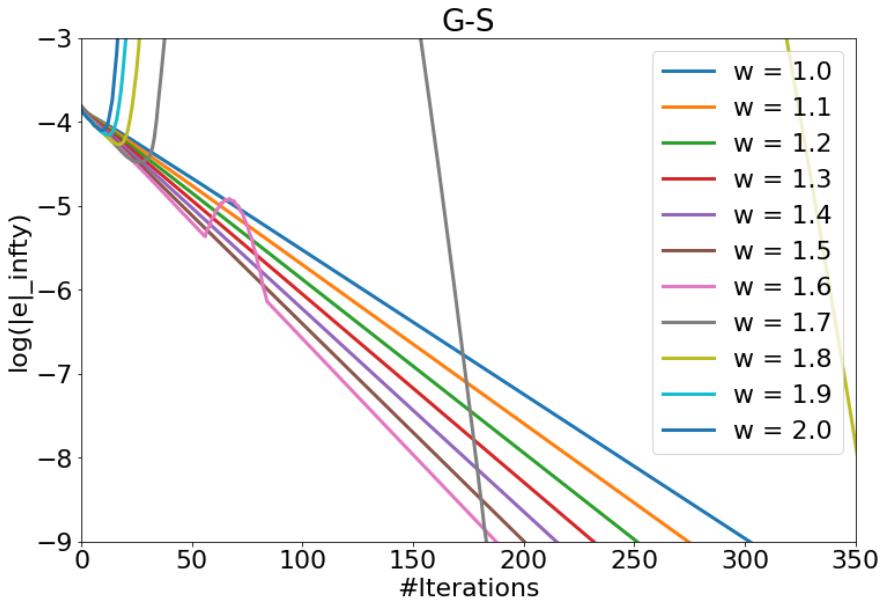
The optimal geometries are listed in the above table. The maximal Q is given by h=1, c=0.5, the one we solved before. These optimal geometry has a common feature that they have close length in the two spatial coordinates (neither too high nor too long).

Problem 2 - Iterative Methods: Jacobi, G-S, Multigrid

1. Jacobi iteration schemes:
   Internal points:
   $$\phi_{i,j}^{t+1} = \frac{1}{4}\left(\phi_{i,j-1}^{t} + \phi_{i,j+1}^{t} + \phi_{i-1,j}^{t} + \phi_{i+1,j}^{t}\right) + \frac{1}{4} f_{ij} \Delta x^2$$
   Boundary points:
   $$\phi_{0,j}^{t+1} = \phi_{N-1,j}^{t+1} = \phi_{i,0}^{t+1} = \phi_{i,N-1}^{t+1} = 0$$

   Gauss-Siedel iteration schemes:
   Internal points:
   We iterate for $i = 0 : N-1$ for $j = 0 : N-1$
   $$\phi_{i,j}^{t+1} = \frac{1}{4}\left(\phi_{i,j-1}^{t+1} + \phi_{i,j+1}^{t} + \phi_{i-1,j}^{t+1} + \phi_{i+1,j}^{t}\right) + \frac{1}{4} f_{ij} \Delta x \Delta y$$
   Boundary points:
   $$\phi_{0,j}^{t+1} = \phi_{N-1,j}^{t+1} = \phi_{i,0}^{t+1} = \phi_{i,N-1}^{t+1} = 0$$
2. a. The equations are shown above, and the implementations are shown in 2.2.py
   b. Convergence plots with several relaxation factors are shown below. 10 relaxation factors are shown for both Jacobi and G-S methods.



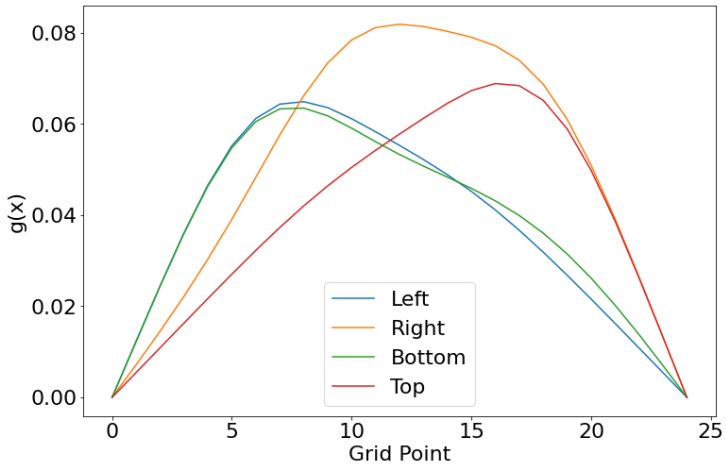


We can see that the optimal relaxation factor for G-S method is $w = 1.6$. Here, we calculate the error by comparing with exact solution.

c. Here we provide the results of normal flux calculated by the G-S method. The results are in good agreement with the sample solution. This is output by 2.2c.py, where the flux is evaluated by forward/backward expression of derivative, giving 2nd order accuracy.

	Left	Right	Bottom	Top
1	0.0	0.0	0.0	0.0
2	0.0123	0.0072	0.0122	0.0054
3	0.0244	0.0145	0.0244	0.0109
4	0.0361	0.0222	0.036	0.0163
5	0.0466	0.0303	0.0464	0.0217
6	0.0552	0.0391	0.0548	0.027
7	0.0612	0.0483	0.0605	0.0323
8	0.0643	0.0576	0.0633	0.0373

9	0.0649	0.0662	0.0635	0.042
10	0.0636	0.0733	0.0618	0.0464
11	0.0611	0.0784	0.0591	0.0504
12	0.0583	0.0811	0.0561	0.0542
13	0.0553	0.0819	0.0533	0.0578
14	0.0522	0.0814	0.0507	0.0612
15	0.0489	0.0803	0.0483	0.0645
16	0.0452	0.079	0.0459	0.0673
17	0.0411	0.0771	0.0431	0.0688
18	0.0367	0.074	0.0399	0.0684
19	0.0319	0.0687	0.036	0.0652
20	0.0268	0.061	0.0315	0.0589
21	0.0216	0.0509	0.0262	0.0498
22	0.0162	0.0392	0.0202	0.0387
23	0.0108	0.0264	0.0138	0.0262
24	0.0054	0.0132	0.007	0.0132
25	0.0	0.0	0.0	0.0



3. We choose the coarse grid with Δx two times of the fine grid. We use restriction method that directly assign the value of the fine grid on the coarse grid to the coarse grid value:

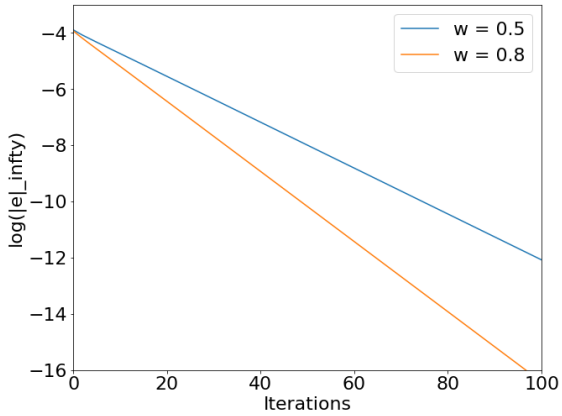
$$r_{i,j}^{coarse} = r_{2i,2j}^{fine}$$

We implement prolongation by representing the value on the coarse grid by the value on the coarse grid, and the value in between the average between values on the adjacent coarse grid:

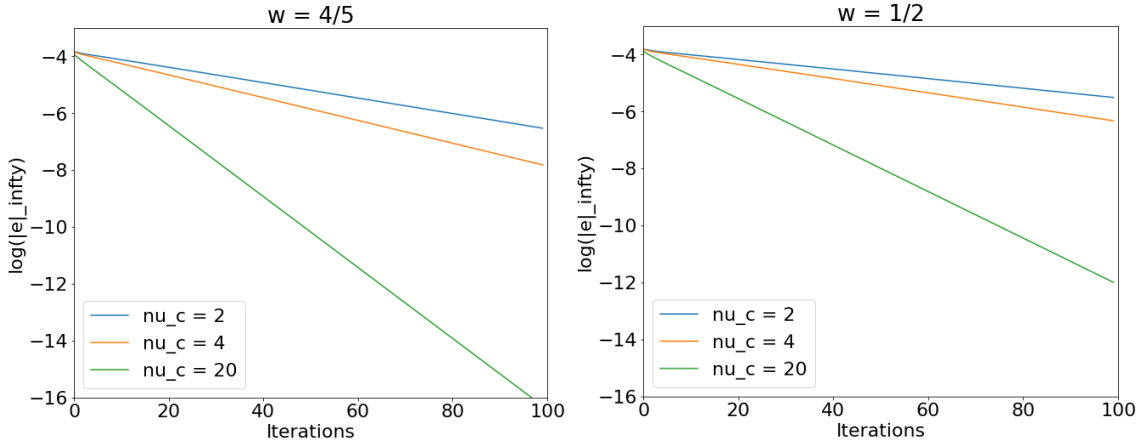
$$e_{2i,2j}^{fine} = e_{i,j}^{coarse}$$
$$e_{2i+1,2j}^{fine} = (e_{i,j}^{coarse} + e_{i+1,j}^{coarse})/2$$
$$e_{2i,2j+1}^{fine} = (e_{i,j}^{coarse} + e_{i,j+1}^{coarse})/2$$
$$e_{2i+1,2j+1}^{fine} = (e_{i,j}^{coarse} + e_{i+1,j}^{coarse} + e_{i,j+1}^{coarse} + e_{i+1,j+1}^{coarse})/4$$

The implementation is in code 2.3.py

(a) The results using multigrid method with Jacobi method is shown below. We have $\nu_1 = \nu_2 = 1$, and $\nu_C = 20$ as a proxy for the exact solution. We can see that the multigrid method converges much faster than both Jacobi and G-S shown in the previous questions. The log(e) converges to -10 after only 50 steps for $w = 0.8$ in multigrid method, while it takes more than 200 steps for the best case of G-S and more than 500 steps for Jacobi on a single grid. The relaxation factor of $w = 4/5$ is better in my multigrid routine.



(b) We implement different values of ν_c as shown below. For exact, we take $\nu_c = 20$. The routine with larger ν_c gives faster convergence with respect to iterations. We can conclude that if we do the coarse mesh problem more accurately, the solution will converge faster. It worth note that, however, if we compare the total number of Jacobi relaxation steps, using larger ν_c may not be advantageous, because that requires more Jacobi relaxation steps per iteration.



4. We implement the search of all possible combinations in codes 2.4.py, and output the one with the smallest difference with the $g(x)$ given in the file. We obtain the positions of the four blocks at the positions:

$$(2,7,10,11)$$

Which gives the smallest difference with

$$\| e \|_\infty = 4.95 \times 10^{-5}$$

The location of sources are shown pictorially as below