Question 1

Section a:

For a vector u denote u_c and u_f to be the first c and f elements of u respectively, so we can write $u = u_c + u_f$, also denote u^* to be the exact solution to the linear equation $Au = b$. So the equation for the error in the m iteration will be:

$$u^* - u^{m+1} = u_c^* - u_c^{m+1} + u_f^* - u_f^{m+1}$$

Using the F-relaxation formula we get:

$$u^* - u^{m+1} = u_c^* - u_c^m + u_f^* - u_f^m - A_{ff}^{-1}(b_f - A_{fc}u_c^m - A_{ff}u_f^m)$$

Now notice that $b_f = A_{fc}u_c^* + A_{ff}u_f^*$. Putting it on the last equation:

$$u^* - u^{m+1} = u_c^* - u_c^m + u_f^* - u_f^m - A_{ff}^{-1}\left(A_{fc}u_c^* + A_{ff}u_f^* - A_{fc}u_c^m - A_{ff}u_f^m\right) =$$
$$u_c^* - u_c^m + u_f^* - u_f^m - I_f\left(u_f^* - u_f^m\right) - A_{ff}^{-1}A_{fc}(u_c^* - u_c^m)$$

Recall the definition of the error vector $e^k = x^* - x^k$:

$$e^{m+1} = e_c^m + e_f^m - I_f e_f^m - A_{ff}^{-1}A_{fc}e_c^m = \left(I_c - A_{ff}^{-1}A_{fc}\right)e_c^m$$

And that finishes the proof since:

$$\begin{bmatrix} I_c & 0 \\ -A_{ff}^{-1}A_{fc} & 0 \end{bmatrix} e^m = \left(I_c - A_{ff}^{-1}A_{fc}\right)e_c^m$$

Section b:

In the two-level cyclic reduction, we only apply the recursive call once. So, according to the algorithm we saw at the lecture, the linear equation passed as parameter to only recursive call is solved by some direct method. The parameters we pass are:

$\boldsymbol{\hat{A}_{cc}} = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$ and $\boldsymbol{\hat{b}_c} = b_c - A_{cf}A_{ff}^{-1}b_f$

So, at the two-level cyclic reduction we solve directly: $\hat{A}_{cc}u = \hat{b}_c$ witch yields:

$$u_c = \hat{A}_{cc}^{-1}\hat{b}_{cc} = \hat{A}_{cc}^{-1}\left(b_c - A_{cf}A_{ff}^{-1}b_f\right) = \hat{A}_{cc}^{-1}\begin{bmatrix} I_c & -A_{cf}A_{ff}^{-1} \end{bmatrix}\begin{bmatrix} b_c \\ b_f \end{bmatrix} = \hat{A}_{cc}{}^{-1}\hat{R}b$$

After calculating u_c , according to the algorithm:

$$u_f = A_{ff}^{-1}(b_f - A_{fc}u_c) = A_{ff}^{-1}\left(b_f - A_{fc}\hat{A}_{cc}{}^{-1}\hat{R}b\right).$$

If we set $u_f^0 = 0$ before the relaxation, then we get after one iteration:

$$u_f^1 = u_f^0 + A_{ff}^{-1}(b_f - A_{fc}u_c^1 - A_{ff}u_f^0) = A_{ff}^{-1}(b_f - A_{fc}u_c^1)$$

So, we see that in fact applying two-level cyclic reduction in our case is equal to applying one $F - relaxation$ when we set $u_f^0 = 0$.

Moreover, we can see that is does not matter what value u_f^0 gets before the relaxation by the definition of F-relaxation:

$$u_f^{k+1} = u_f^k + A_{ff}^{-1}(b_f - A_{fc}u_c^k - A_{ff}u_f^k) = u_f^k + A_{ff}^{-1}(b_f - A_{fc}u_c^k) - A_{ff}^{-1}A_{ff}u_f^k =$$
$$= A_{ff}^{-1}(b_f - A_{fc}u_c^k) + \underbrace{u_f^k - I_f u_f^k}_{=0} = A_{ff}^{-1}(b_f - A_{fc}u_c^k)$$

We received that the value of the next iteration u_f^{k+1} is independent from u_f^k. And we also get: $u_f^1 = u_f^0 + A_{ff}^{-1}(b_f - A_{fc}u_c^1 - A_{ff}u_f^0) = A_{ff}^{-1}(b_f - A_{fc}u_c^1)$ even if $u_f^0 \neq 0$.

Section c:

We will use the notations of the iterative two level grid scheme that appears in the lecture notes, following the algorithm step by step in order to analyze the iteration matrix.

Our iterative scheme solves $A^h u = b^h$, suppose we are at the n, so to get u^{n+1} from u^n the following steps occurring:

<u>Step 1:</u> Smoothing the error with v_1 relaxations. Denote v^n to be the approximate solution after the relaxations, and e^k to be the error only for the relaxations. So $e^{v_1} = S^{v_1}e^0$, and since $e^0 = u^* - u^n$ we get $v^n = u^* - e^{v_1} = u^* - S^{v_1}e^0 = u^* - S^{v_1}(u^* - u^n)$.

<u>Step 2:</u> Project $r = b^h - A^h v^n$ onto Ω^{2h} to define b^{2h}. So using step 1 calculations we get: $b^{2h} = I_h^{2h}r = I_h^{2h}(b^h - A^h v^n) = I_h^{2h}(A^h u^* - A^h(u^* - S^{v1}(u^* - u^n))) =$

$$= I_h^{2h}A^h S^{v_1}(u^* - u^n)$$

Step 3: Defining e^{2h} to be the solution to the coarse equation $A^{2h}e^{2h} = b^{2h}$ which means that $e^{2h} = (A^{2h})^{-1}b^{2h} = (A^{2h})^{-1}I_h^{2h}A^hS^{v_1}(u^* - u^n)$

Step 4: Transferring the coarse grid solution e^{2h} to the fine grid as e^h. Which means that: $e^h = I_{2h}^h e^{2h} = I_{2h}^h(A^{2h})^{-1}I_h^{2h}A^hS^{v_1}(u^* - u^n)$.

Step 5: Denote $w^n = v^n + e^h$, to be the correction of the fine grid solution.

Step 6: Defining u^{n+1} to be w^n after v_2 post-relaxations.

We can now derive the iteration matrix:

$$\begin{aligned} u^* - w^n = u^* - v^n - e^h &= u^* - u^* + S^{v_1}e^{(n)} - I_{2h}^h(A^{2h})^{-1}I_h^{2h}A^hS^{v_1}e^{(n)} \\ &= \left(I - I_{2h}^h(A^{2h})^{-1}I_h^{2h}A^h\right)S^{v_1}e^n \end{aligned}$$

Applying step 6 we conclude: $e^{(n+1)} = S^{v_2}\left(I - I_{2h}^h(A^{2h})^{-1}I_h^{2h}A^h\right)S^{v_1}e^n$ which completes the proof.

Section d:

Assuming that $S^{v_1}e^k$ is in the range of the prolongation, there exist a vector at the coarse grid e^{2h} such that $S^{v_1}e^k = I_{2h}^h e^{2h}$. Putting it in the iteration matrix we found at section c we get:

$$\begin{aligned} &e^{(k+1)} = S^{v_2}\left(I - I_{2h}^h(A^{2h})^{-1}I_h^{2h}A^h\right)S^{v_1}e^k = S^{v_2}\left(I - I_{2h}^h(A^{2h})^{-1}I_h^{2h}A^h\right)I_{2h}^h e^{2h} = \\ &= S^{v_2}\left(I_{2h}^h e^{2h} - I_{2h}^h(A^{2h})^{-1}I_h^{2h}A^hI_{2h}^h e^{2h}\right) = S^{v_2}\left(I_{2h}^h e^{2h} - I_{2h}^h(A^{2h})^{-1}A^{2h}e^{2h}\right) = \\ &S^{v_2}\left(I_{2h}^h e^{2h} - I_{2h}^h e^{2h}\right) = 0 \end{aligned}$$

That means that under the assumption of $S^{v_1}e^k = I_{2h}^h e^{2h}$ together with using the Galerkin operator, the error is eliminating to zero.

Section e:

Question 2

The smoothing factor is defined by $\mu(S^h) = \max\limits_{\frac{\pi}{2}<|\theta_i|<\pi} \{|\hat{S}_h(\vec{\theta})|\}$

Also, we saw that $\hat{S}^h_{Jac}$ for the 2D passion equation is - $\frac{(cos\theta_1 + cos\theta_2)}{2}$

Like we studied in class, to compute the optimal damping factor, we will want to minimize $\mu(\hat{S}_{\omega-Jac})$ with respect to ω. That means that for the 2D poisson equation we want to solve the following equation:

$$\mu(\hat{S}^h_{\omega-Jac}) = (1-\omega) + \omega\hat{S}^h_{Jac} = (1-\omega) + \frac{\omega(cos\theta_1 + cos\theta_2)}{2}$$

The maximum value is obtained by equalising the extrems of $\hat{S}^h$ at the oscillatory modes So like we studied in class we want to solve : $\left|1-\omega+\omega\hat{S}^h_{min}\right| = \left|1-\omega+\omega\hat{S}^h_{max}\right|$

under the range is $\frac{\pi}{2} < |\theta_i| < \pi$. Then we can conclude that the minimum and maximum values of $\mu(\hat{S}^h_{\omega-Jac})$ are:

Minimum: $\hat{S}^h_{\omega-Jac} = (1-\omega) + \omega\hat{S}^h_{Jac}\left(\frac{\pi}{2}, \pi\right) = 1 - \frac{\omega}{2}$

Maximum: $\hat{S}^h_{\omega-Jac} = (1-\omega) + \omega\hat{S}^h_{Jac}(\pi, \pi) = 1 - 2\omega$

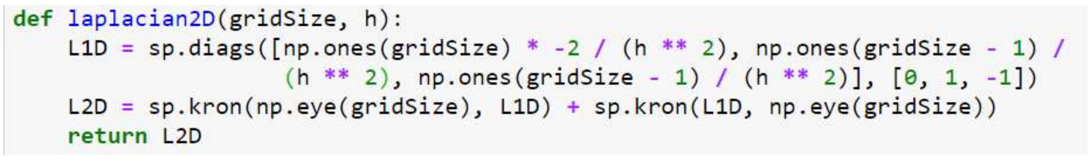
So putting all togther we get: $|1-2\omega| = \left|1-\frac{\omega}{2}\right|$ and there for the solution is:

$$2\omega - 1 = 1 - \frac{\omega}{2} \Rightarrow \boldsymbol{\omega_{opt}} = \frac{\mathbf{4}}{\mathbf{5}}$$

Question 3

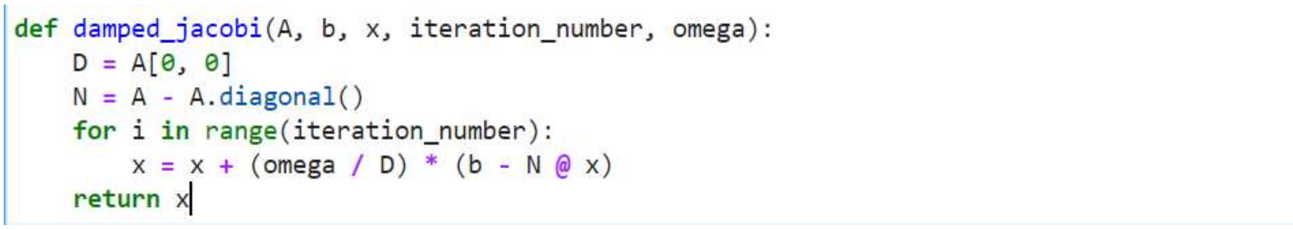
Section a:

- Function that returns the Laplacian operator:



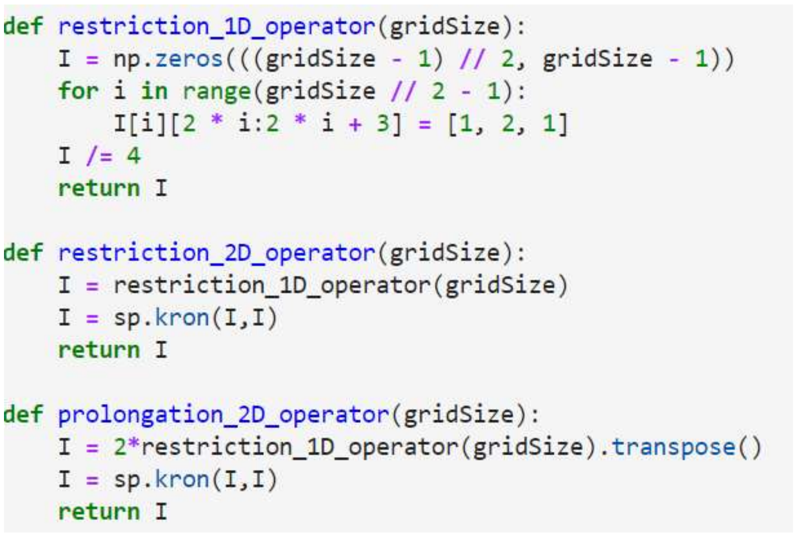
```
def laplacian2D(gridSize, h):
    L1D = sp.diags([np.ones(gridSize) * -2 / (h ** 2), np.ones(gridSize - 1) /
                    (h ** 2), np.ones(gridSize - 1) / (h ** 2)], [0, 1, -1])
    L2D = sp.kron(np.eye(gridSize), L1D) + sp.kron(L1D, np.eye(gridSize))
    return L2D
```

- A function for applying the damped Jacobi iteration:



```
def damped_jacobi(A, b, x, iteration_number, omega):
    D = A[0, 0]
    N = A - A.diagonal()
    for i in range(iteration_number):
        x = x + (omega / D) * (b - N @ x)
    return x
```

- Two functions for applying the full-weighting restriction and bilinear prolongation to a fine and coarse vectors respectively:



```
def restriction_1D_operator(gridSize):
    I = np.zeros(((gridSize - 1) // 2, gridSize - 1))
    for i in range(gridSize // 2 - 1):
        I[i][2 * i:2 * i + 3] = [1, 2, 1]
    I /= 4
    return I

def restriction_2D_operator(gridSize):
    I = restriction_1D_operator(gridSize)
    I = sp.kron(I,I)
    return I

def prolongation_2D_operator(gridSize):
    I = 2*restriction_1D_operator(gridSize).transpose()
    I = sp.kron(I,I)
    return I
```

- Function for multilevel cycle which receives the parameters:

  Matrix **A** (2D-laplacian operator), solution vector **b**, initial guess **u**, number of levels **level**, size of the grid **gridSize**, number of smoothing iterations **smoothing_iterations**, weight for Jacobi iterations omega.

```
def multilevel_cycle(A, u, b, level, gridSize, smoothing_iterations, omega):
    if level == 0 or b.shape[0] < 4:
        return sp.linalg.spsolve(A, b)

    # pre-relaxation
    u = damped_jacobi(A, b, u, smoothing_iterations, omega)

    residual = b - A @ u
    R = restriction_2D_operator(gridSize)
    P = prolongation_2D_operator(gridSize)
    A_2h = R @ A @ P
    b_2h = R @ residual
    coarse_gridSize = (gridSize + 1) // 2
    u_2h = np.zeros((coarse_gridSize-1)**2)
    e_2h = multilevel_cycle(A_2h, u_2h, b_2h, level-1, coarse_gridSize, smoothing_iterations, omega)
    e_h = P @ e_2h
    u += e_h

    # post-relaxations
    u = damped_jacobi(A, b, u, smoothing_iterations, omega)
    return u
```

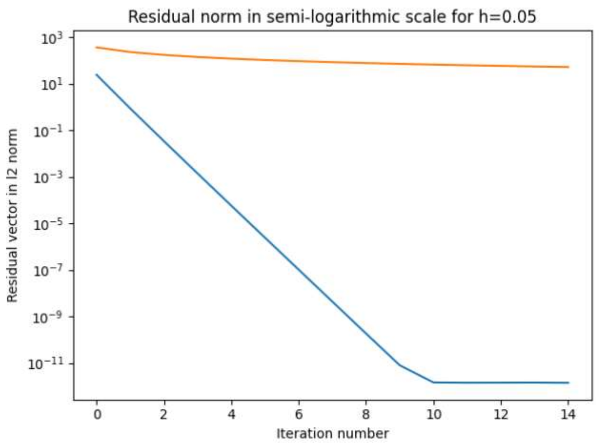
Section b:

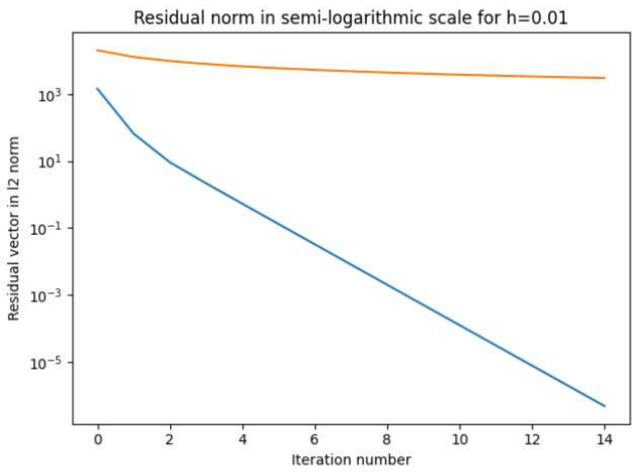
We will now demonstrate our code on the problem defined on the assignment. We will define grids with $h = 0.05, 0.01$, applying 15 iterations of multilevel cycle with $v_1 = v_2 = 1$ relaxation each iteration and $\omega = 0.8$ for damped Jacobi. In other words, for each h we run the following:

```
for i in range(15):
    u = multilevel_cycle(-lap2D, u, b, -1, gridSize, 1, 0.8)
    residual_vector.append(np.linalg.norm(u - b))
    error_vector.append(np.linalg.norm(u-actual_function))
```

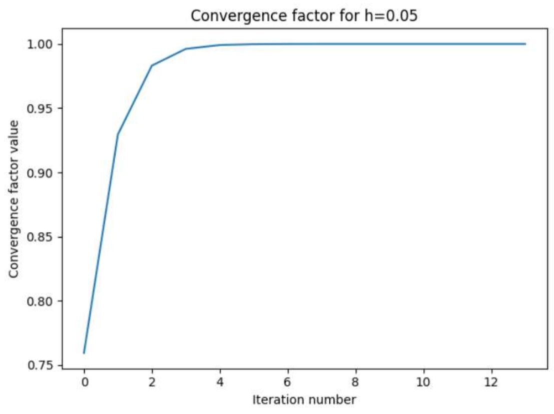
Full code for how we defined **b** with Derichlet boundary conditions and **actual_function** can we found in zip file.

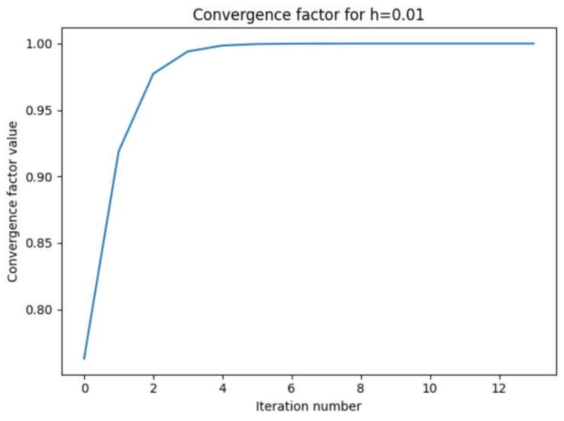
Graphs of the residual norm w.r.t iterations, comparing one multilevel iteration (blue) with five Jacobi iterations (orange):



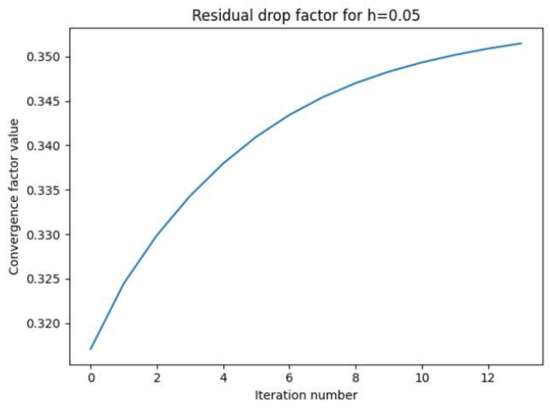


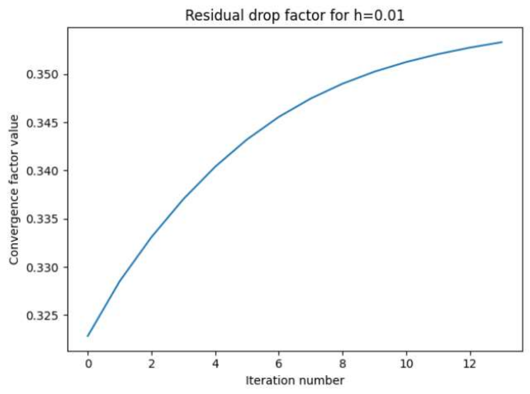
Convergence factor:



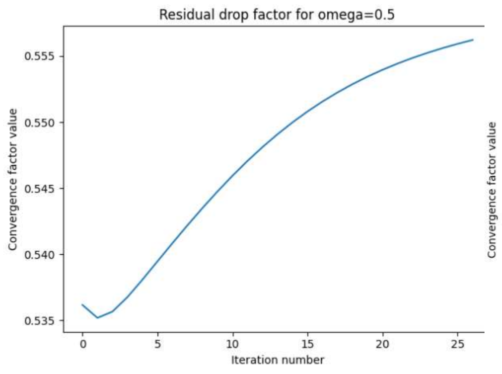


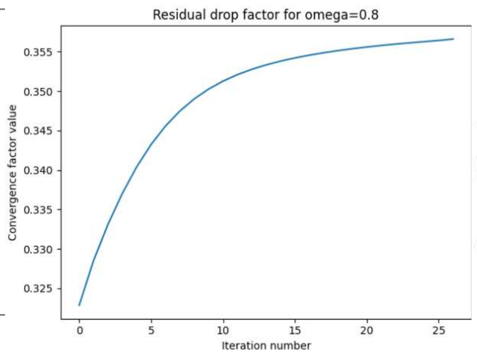
Residual drop factor:

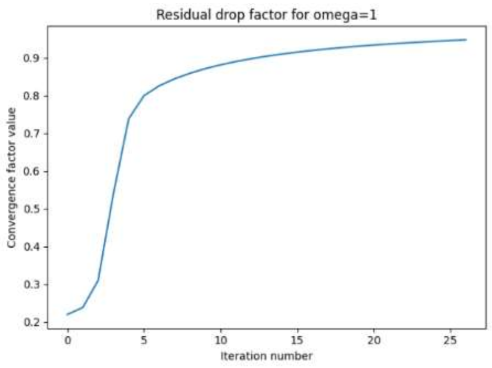




Now we will add more iterations of multigrid in order to numerically compute the residual drop factor for different ω. We will compare $\omega = 0.5, 0.8, 1$ to see if our results are consistent with the theoretical proof of question 2.







In the plots we can see that for $\omega = 0.5$ the residual drop factor is around 0.5 for all the iterations. For $\omega = 1$ the residual drop factor is stabializing around 0.9, and for $\omega = 0.8$ approximatly 0.355.

These results are consistent with the theoretical proof of question 2, since low residual factor means we keep reducing the residual in reletively large amaount even after 25 iterations.