MA5232: Assignment 1

Rahul Venkatesh

March 1, 2024

Contents

1 Problem

Consider the following 3D equation defined on $\Omega = (-1,1) \times (-1,1) \times (-1,1)$.

$$\frac{\partial F_1}{\partial x} = \sigma(\frac{1}{6}\Sigma F_i - F_1)$$

$$-\frac{\partial F_2}{\partial x} = \sigma(\frac{1}{6}\Sigma F_i - F_2)$$

$$\frac{\partial F_3}{\partial y} = \sigma(\frac{1}{6}\Sigma F_i - F_3)$$

$$-\frac{\partial F_4}{\partial y} = \sigma(\frac{1}{6}\Sigma F_i - F_4)$$

$$\frac{\partial F_5}{\partial z} = \sigma(\frac{1}{6}\Sigma F_i - F_5)$$

$$-\frac{\partial F_6}{\partial z} = \sigma(\frac{1}{6}\Sigma F_i - F_6)$$

The boundary conditions are

$$\begin{aligned}
F_1(-1,y,z) &= F_b(y,z) \\
F_3(x,-1,z) &= F_b(x,z) \\
F_5(x,y,-1) &= F_b(x,y) \\
F_2(1,y,z) &= 0 \\
F_4(x,1,z) &= 0 \\
F_6(x,y,1) &= 0
\end{aligned}$$

where

$$F_b(p,q) = \begin{cases} 1 & \text{if } |p| \le 0.2 \text{ and } |q| \le 0.2 \\ 0 & \text{otherwise} \end{cases}$$

Solve the system of equations numerically for $\sigma = 0.1, 1, 10, 100$.

2 Methods

Iterative methods are used to solve the system of PDEs. Starting with an initial guess of the solution, in each iteration, the solution is improved using the solution from the previous iteration. The process is continued until the solution converges. The convergence criterion is that the computed residuals are as little as the set threshold.

The derivatives are approximated with finite differences, and the interval Ω is discretized into a uniform grid with $\Delta x = \Delta y = \Delta z = \frac{2}{N}$ where N is the number of cells. Consider point $p_i = -1 + i\frac{\Delta p}{2}$. The 1-indexed $cell[i, j, k]$ represents the region

$$T_{ijk} = [x_i - \frac{\Delta x}{2}, x_i + \frac{\Delta x}{2}] \times [y_i - \frac{\Delta y}{2}, y_i + \frac{\Delta y}{2}] \times [z_i - \frac{\Delta z}{2}, z_i + \frac{\Delta z}{2}]$$

For simplicity, rename the functions F_1 through F_6 as a, b, c, d, e and f respectively. Approximate $a(x, y, z)$ in the region $cell[i, j, k]$ as

$$A_{ijk} \approx \frac{1}{|T_{ijk}|} \int_{T_{ijk}} a(x, y, z) dxdydz$$

Similarly, define approximations of the other 5 functions.

Obtain the following system of linear equations using finite differences to approximate the partial derivatives.

$$\begin{aligned}
\frac{A_{ijk} - A_{i-1,jk}}{\Delta x} &= \sigma[\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - A_{ijk}] \\
\frac{B_{ijk} - B_{i+1,jk}}{\Delta x} &= \sigma[\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - B_{ijk}] \\
\frac{C_{ijk} - C_{i,j-1,k}}{\Delta y} &= \sigma[\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - C_{ijk}] \\
\frac{D_{ijk} - D_{i,j+1,k}}{\Delta y} &= \sigma[\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - D_{ijk}] \\
\frac{E_{ijk} - E_{ij,k-1}}{\Delta z} &= \sigma[\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - E_{ijk}] \\
\frac{F_{ijk} - F_{ij,k+1}}{\Delta z} &= \sigma[\frac{1}{6}(A_{ijk} + B_{ijk} + C_{ijk} + D_{ijk} + E_{ijk} + F_{ijk}) - F_{ijk}]
\end{aligned}$$

with suitable boundary conditions by converting points in the domain into indices in the grid.

2.1 Fixed-point Iteration (FP)

The idea is to solve each of the six equations individually assuming the other variables are given.

$$
\begin{aligned}
\frac{A_{ijk}^{(l+1)} - A_{i-1,jk}^{(l+1)}}{\Delta x} &= \sigma[\frac{1}{6}(A_{ijk}^{(l+1)} + B_{ijk}^{(l)} + C_{ijk}^{(l)} + D_{ijk}^{(l)} + E_{ijk}^{(l)} + F_{ijk}^{(l)}) - A_{ijk}^{(l+1)}] \\
\frac{B_{ijk}^{(l+1)} - B_{i+1,jk}^{(l+1)}}{\Delta x} &= \sigma[\frac{1}{6}(A_{ijk}^{(l)} + B_{ijk}^{(l+1)} + C_{ijk}^{(l)} + D_{ijk}^{(l)} + E_{ijk}^{(l)} + F_{ijk}^{(l)}) - B_{ijk}^{(l+1)}]
\end{aligned}
$$

where $A^{(l)}$ is A in the l^{th} iteration.

In the first equation, A is computed using B, C, D, E and F from the previous iteration. Similarly, in the second equation, B is computed using A, C, D, E and F from the previous iteration. One could also use latest values computed in the current iteration. For example, in the second equation, $A^{(l+1)}$ can be used instead of $A^{(l)}$. The values can be updated linearly. Note that B needs to be updated backwards and A forwards.

The update rules follow for the rest of the matrices in y and z dimensions.

2.1.1 Relaxation

Each iteration only contains six small loops (of size N) and matrix additions making the iterations quite fast (around 5 iterations per second). But the problem arises when σ is large: while the individual iterations are fast, convergence is slow. For $\sigma = 100$, after 1000 iterations, the residual was still 1.8.

To solve this slow convergence issue, each update is relaxed with an additional term.

$$
\begin{aligned}
\epsilon(A_{ijk}^{(l+1)} - A_{ijk}^{(l)}) + \frac{A_{ijk}^{(l+1)} - A_{i-1,jk}^{(l+1)}}{\Delta x} &= \\
&\sigma[\frac{1}{6}(A_{ijk}^{(l+1)} + B_{ijk}^{(l)} + C_{ijk}^{(l)} + D_{ijk}^{(l)} + E_{ijk}^{(l)} + F_{ijk}^{(l)}) - A_{ijk}^{(l+1)}]
\end{aligned}
$$

However, the choice of ϵ is unclear and the process may diverge with the wrong ϵ.

2.2 Symmetric Gauss-Seidel (SGS)

Each iteration contains two scans. The first scan is carried out in a left-to-right, bottom-to-top, back-to-front fashion; the second scan is carried out in a right-to-left, top-to-bottom, front-to-back fashion.

First scan:

$$\frac{A_{ijk}^{(l+1)} - A_{i-1,jk}^{(l+1)}}{\Delta x} = \sigma[\frac{1}{6}(A_{ijk}^{(l+1)} + B_{ijk}^{(l+1)} + C_{ijk}^{(l+1)} + D_{ijk}^{(l+1)} + E_{ijk}^{(l+1)} + F_{ijk}^{(l+1)}) - A_{ijk}^{(l+1)}]$$
$$\frac{B_{ijk}^{(l+1)} - B_{i+1,jk}^{(l)}}{\Delta x} = \sigma[\frac{1}{6}(A_{ijk}^{(l+1)} + B_{ijk}^{(l+1)} + C_{ijk}^{(l+1)} + D_{ijk}^{(l+1)} + E_{ijk}^{(l+1)} + F_{ijk}^{(l+1)}) - B_{ijk}^{(l+1)}]$$

Note that the right-hand side, in contrast to the fixed-point iteration updates, uses the values from current iteration. On the left-hand side, $B_{i+1,jk}$ uses the previous iteration's value as it hasn't been computed in the current scan yet, and $A_{i-1,jk}$ uses the current iteration's value which was computed in the previous step. Similarly, the second scan will use $A_{i-1,jk}^{(l)}$ and $B_{i+1,jk}^{(l+1)}$. Either way, they are both known values. However, the entire right-hand side is unknown. Moving the unknowns to the left gives us the following system of linear equations.

$$M \begin{pmatrix} A_{ijk} \\ B_{ijk} \\ C_{ijk} \\ D_{ijk} \\ E_{ijk} \\ F_{ijk} \end{pmatrix} = x_{ijk} = \begin{pmatrix} \frac{1}{\Delta x} A_{i-1,jk} \\ \frac{1}{\Delta x} B_{i+1,jk} \\ \frac{1}{\Delta y} C_{i,j-1,k} \\ \frac{1}{\Delta y} D_{i,j+1,k} \\ \frac{1}{\Delta z} E_{i,j,k-1} \\ \frac{1}{\Delta z} F_{i,j,k+1} \end{pmatrix}$$

where

$$\begin{aligned} M &= \begin{bmatrix} \frac{1}{\Delta x} & & & & & \\ & \frac{1}{\Delta x} & & & & \\ & & \frac{1}{\Delta y} & & & \\ & & & \frac{1}{\Delta y} & & \\ & & & & \frac{1}{\Delta z} & \\ & & & & & \frac{1}{\Delta z} \end{bmatrix} + \sigma I - \frac{\sigma}{6}\mathbf{1}\mathbf{1}^T \\ &= (\frac{1}{\Delta x} + \sigma)I - \frac{\sigma}{6}\mathbf{1}\mathbf{1}^T \end{aligned}$$

This can be solved for each (i, j, k) by computing the constant M^{-1} and dot product.

```python
for i in range(1, N + 1):
    for j in range(1, N + 1):
        for k in range(1, N + 1):
            x = (N/2) * np.array([
                A[i - 1, j, k],
                B[i + 1, j, k],
                C[i, j - 1, k],
                D[i, j + 1, k],
                E[i, j, k - 1],
                F[i, j, k + 1],
            ])
            A[i, j, k], B[i, j, k], C[i, j, k], \
            D[i, j, k], E[i, j, k], F[i, j, k] = M_inv.dot(x)
```

Listing 1: First Scan

2.2.1 Red-Black ordering

Although the SGS method results in a clean exact solution, the algorithm itself is inherently sequential. Each iteration has two for-loops of size N^3 which makes the updates very slow (around 14 seconds per iteration) and difficult to parallelize (i.e., exploit fast matrix operations).

The idea of the red-black ordering for two-dimensional grid is to color odd grid cells ($i + j \equiv 1 \pmod 2$) red and the even grid cells black. Since the values of the red cells only depend on the black cells and vice versa, the first scan can update all red cells, and the second scan updates all black cells. The main advantage is that the updates can be parallelized.

For this problem, the 3D grid can be colored using three colors: red, blue and green depending on $(i + j + k) \bmod 3$. However, for this problem, this scheme is exceedingly slow in convergence and also slow in each iteration due to the $O(N^3)$ matrix for parallel update.

2.2.2 Successive Over Relaxation (SOR)

For large σ, the convergence is still slow: for $\sigma = 100$, after 1000 iterations, the residual was still 0.02. In the spirit of the Fixed-point iteration method with relaxation, a relaxation factor, ω, can be introduced over here as well. If $\overline{A}_{ijk}^{(l+1)}$ were the original SGS value, the relaxed SGS update would be

$$A_{ijk}^{(l+1)} \leftarrow (1 - \omega) A_{ijk}^{(l)} + \omega \overline{A}_{ijk}^{(l+1)}$$

2.2.3 Optimization

As noted earlier, SGS iterations are time-consuming. Even if SOR helps with faster convergence, just 100 iterations takes 25 minutes. Although the algorithm is still sequential, writing in terms of matrices would be helpful as python's `numpy` provides vectorization benefits.

In the inner most loop, only E depends on the value computed by the previous for-iteration. For given (i, j), compute E_{ijk} (dropping $_{ij}$ for brevity) first as

$$E_k = M_5^{-1}[-5]x_k[-5] + M_{55}^{-1}E_{k-1}$$

where $M_5^{-1}[-5]$ is the 5th row without the 5th column, $x_k[-5]$ is the column vector without the 5th entry (row) and M_{55} is the 5th row and 5th column. This recursion can be converted into element-wise `np.array` multiplications and `np.cumsum`. Now, that all E_k has been computed for given (i, j), the other matrices at all k can be computed by one matrix multiplication as $M^{-1}X$ where $X = [x_1, x_2, \dots x_k, \dots, x_N]$.

This gives us the much needed 15x speed boost: each iteration runs under a second.

3 Setting

Number of points: $N = 100$; threshold of convergence: $\delta = 10^{-6}$.

parameter	$\sigma = 0.1$	$\sigma = 1$	$\sigma = 10$	$\sigma = 100$
SGS ω	1	1	1.05	1.37
FP ϵ	0	0	-3	-40

Table 1: Recommended settings for fast convergence

4 Results

Method	Num of iterations				Time (in s)			
	$\sigma = 0.1$	$\sigma = 1$	$\sigma = 10$	$\sigma = 100$	$\sigma = 0.1$	$\sigma = 1$	$\sigma = 10$	$\sigma = 100$
SGS	4	11	167	1000[1]	3.22	8.92	144.93	844.38[1]
SGS with ω	4	11	47	156	3.90	10.32	43.15	141.30
FP	5	16	328	1000[2]	1.28	3.88	78.26	235.59[2]
FP with ϵ	5	16	96	559	1.22	3.85	20.14	121.76

Table 2: Performance comparison of the methods

[1]did not converge; residual=0.02
[2]did not converge; residual=1.83

4.1 Plots

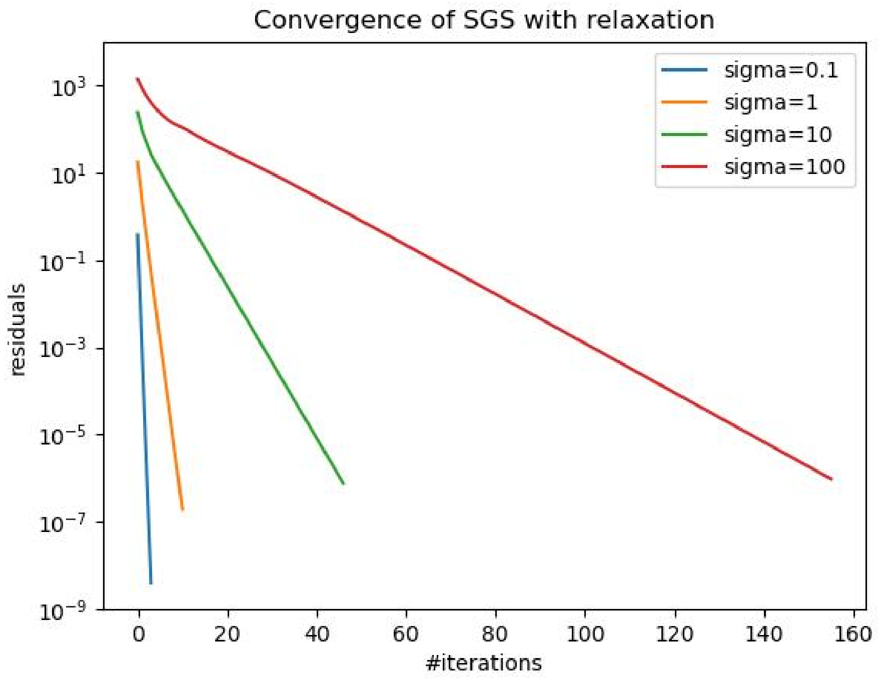
The following pages contain the visual representation of the solution as provided by the SGS method with appropriate relaxations. The solution provided by FP and SGS have RMSE $= 0.002$ (sanity check: basically the same).

Figure 1 shows the residual as the method iterates. The residuals (y-axis) are represented on the log-scale. Both methods yield similar convergence steps except the number of iterations to converge.

Figure 2 and 3 is a heatmap of $(A + B + C + D + E + F)$ plotted on the planes $x = 0, y = 0$ and $z = 0$.

Figure 4 through 7 shows the heatmaps of A, B and $(A + B + C + D + E + F)$ but as a 2D plot instead of a 3D plot.

Figure 1: Residual vs iterations for SGS



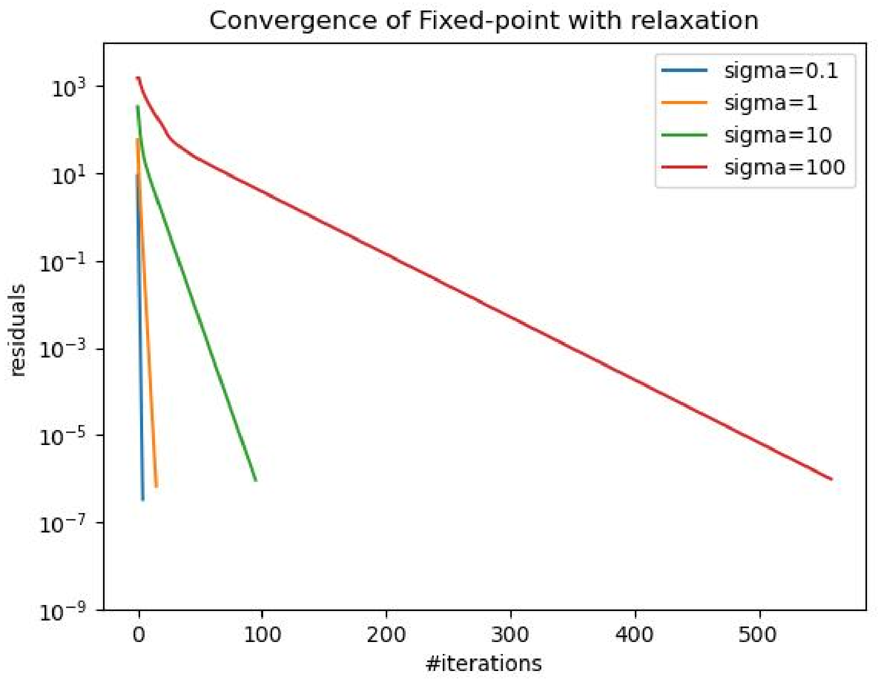
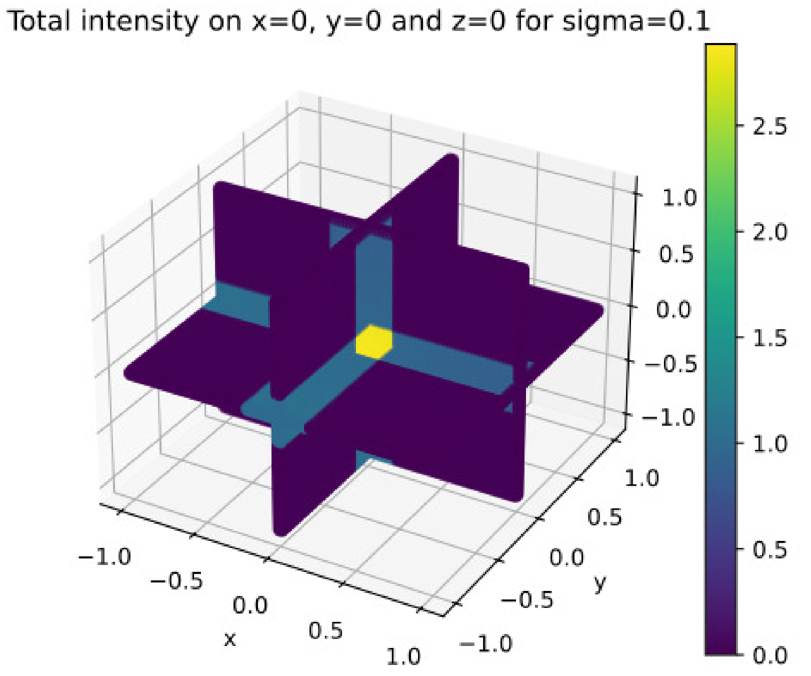


Figure 2: 3D visualization of sum of F_i



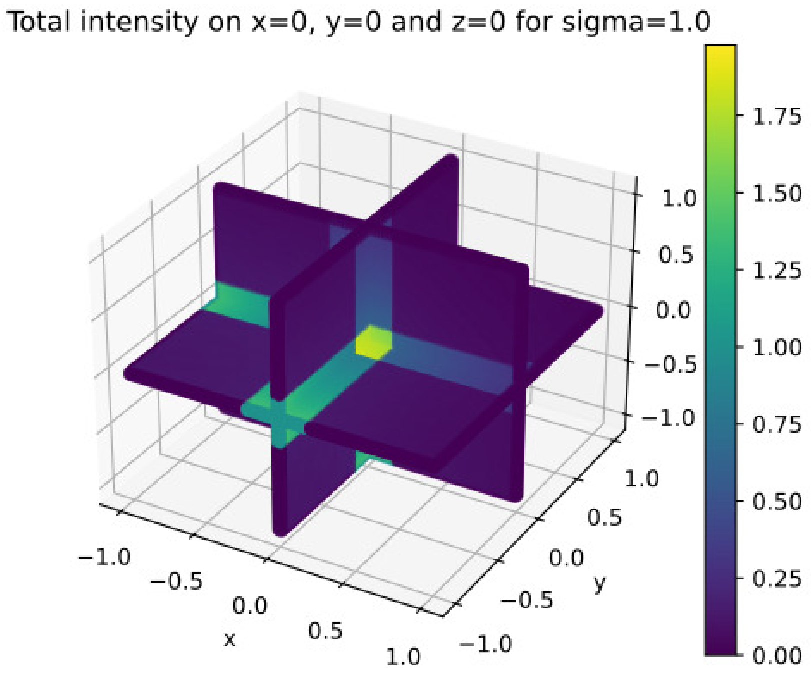
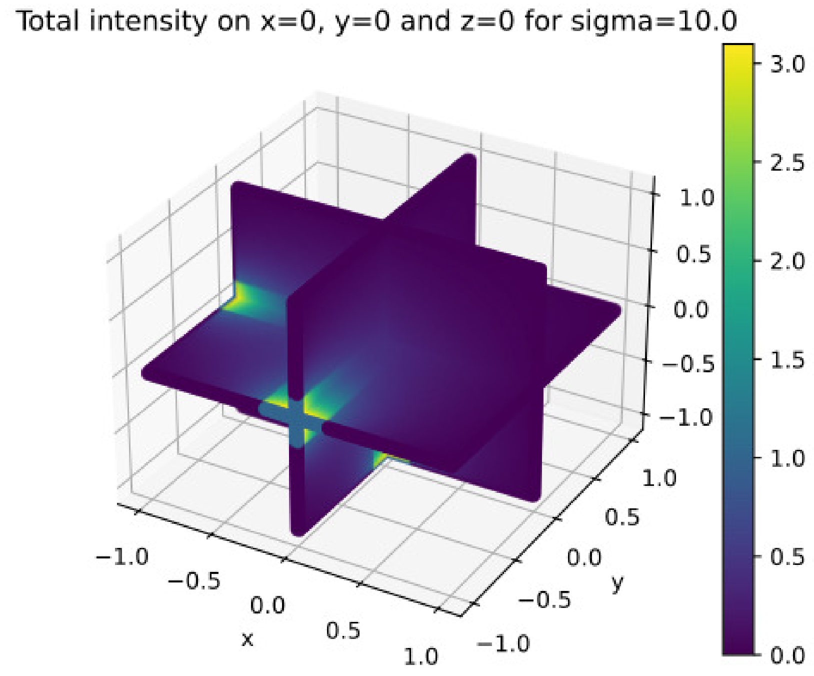


Figure 3: 3D visualization of sum of F_i



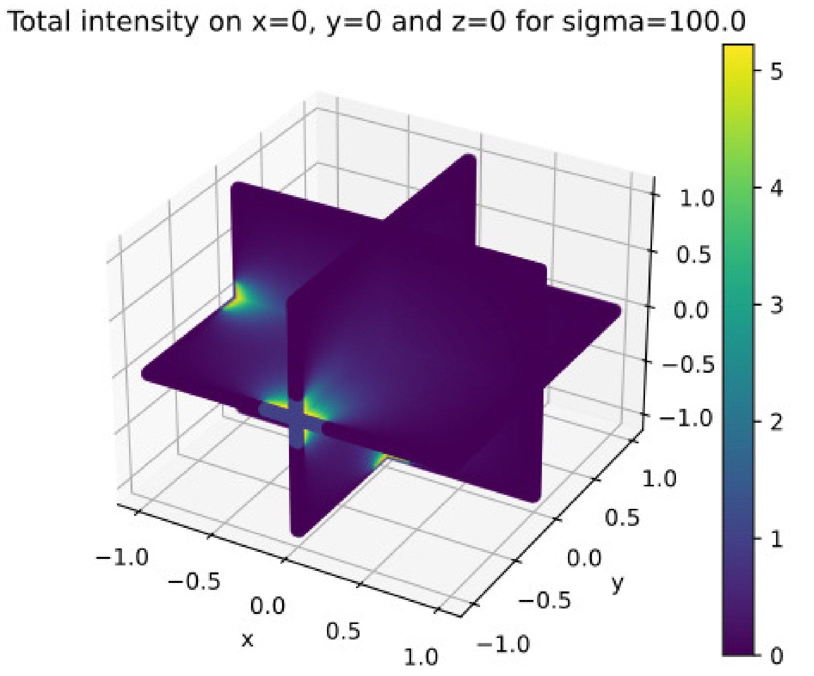


Figure 4: SGS solution for $\sigma = 0.1$

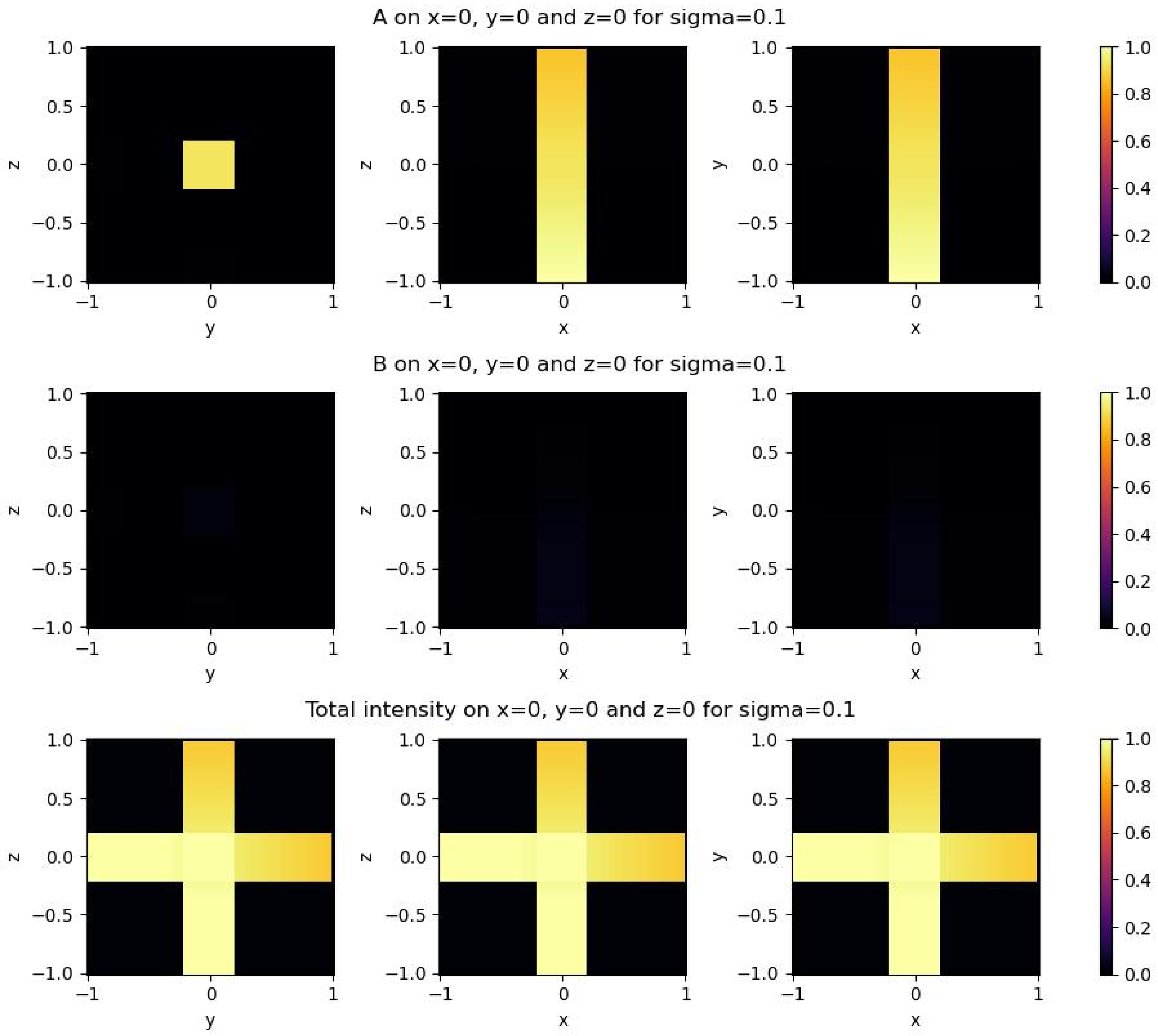


Figure 5: SGS solution for $\sigma = 1$

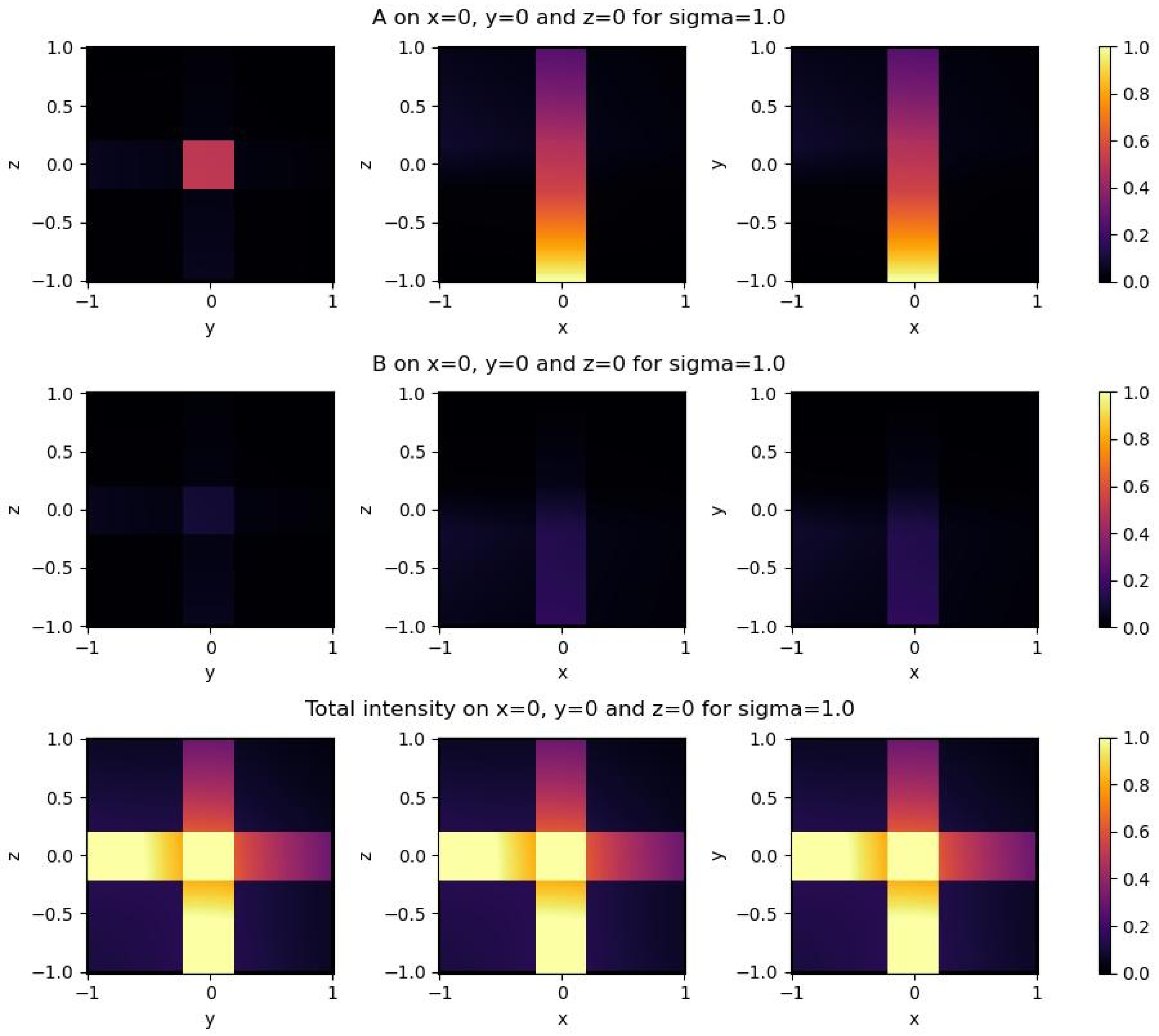


Figure 6: SGS solution for $\sigma = 10$

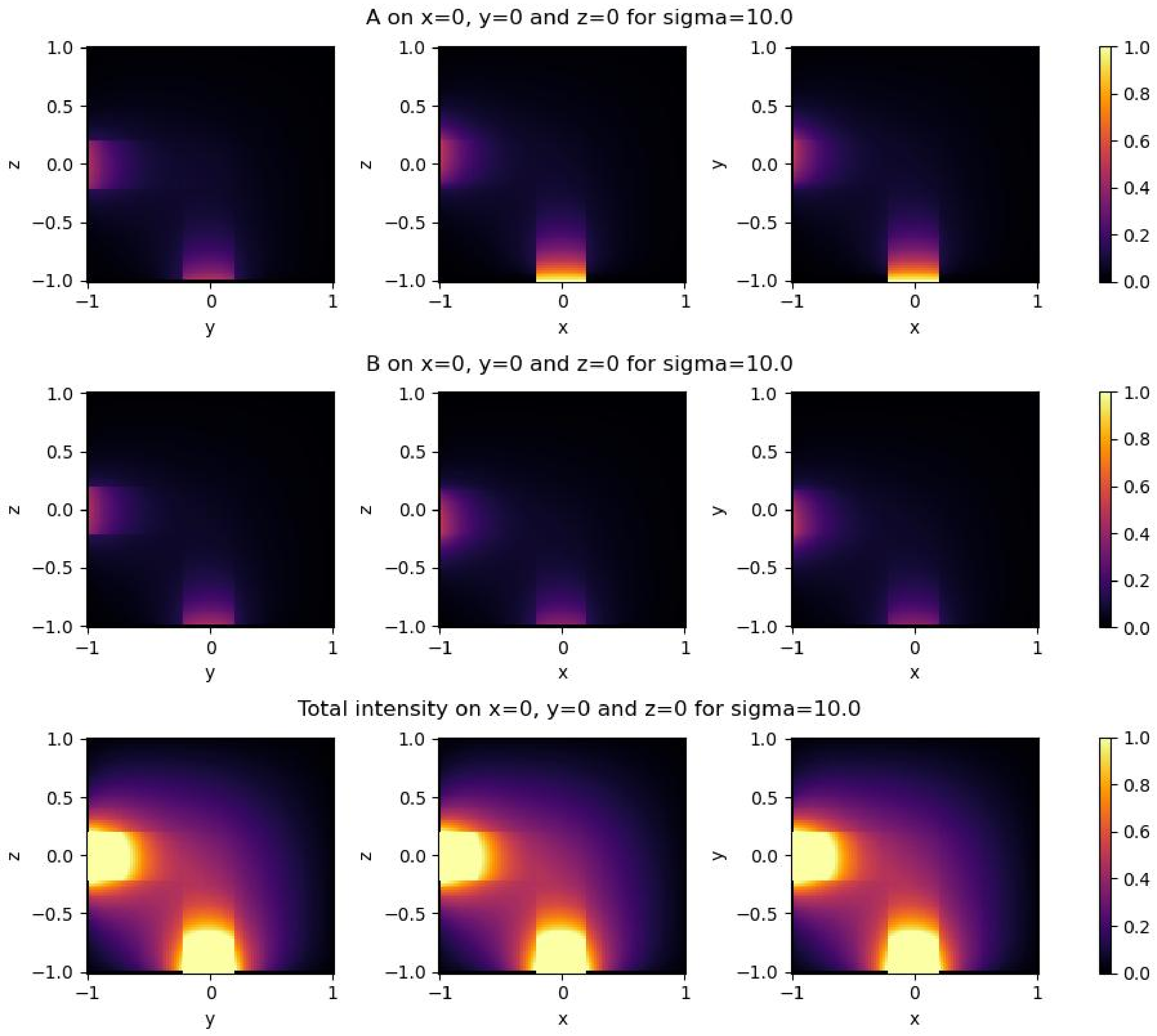
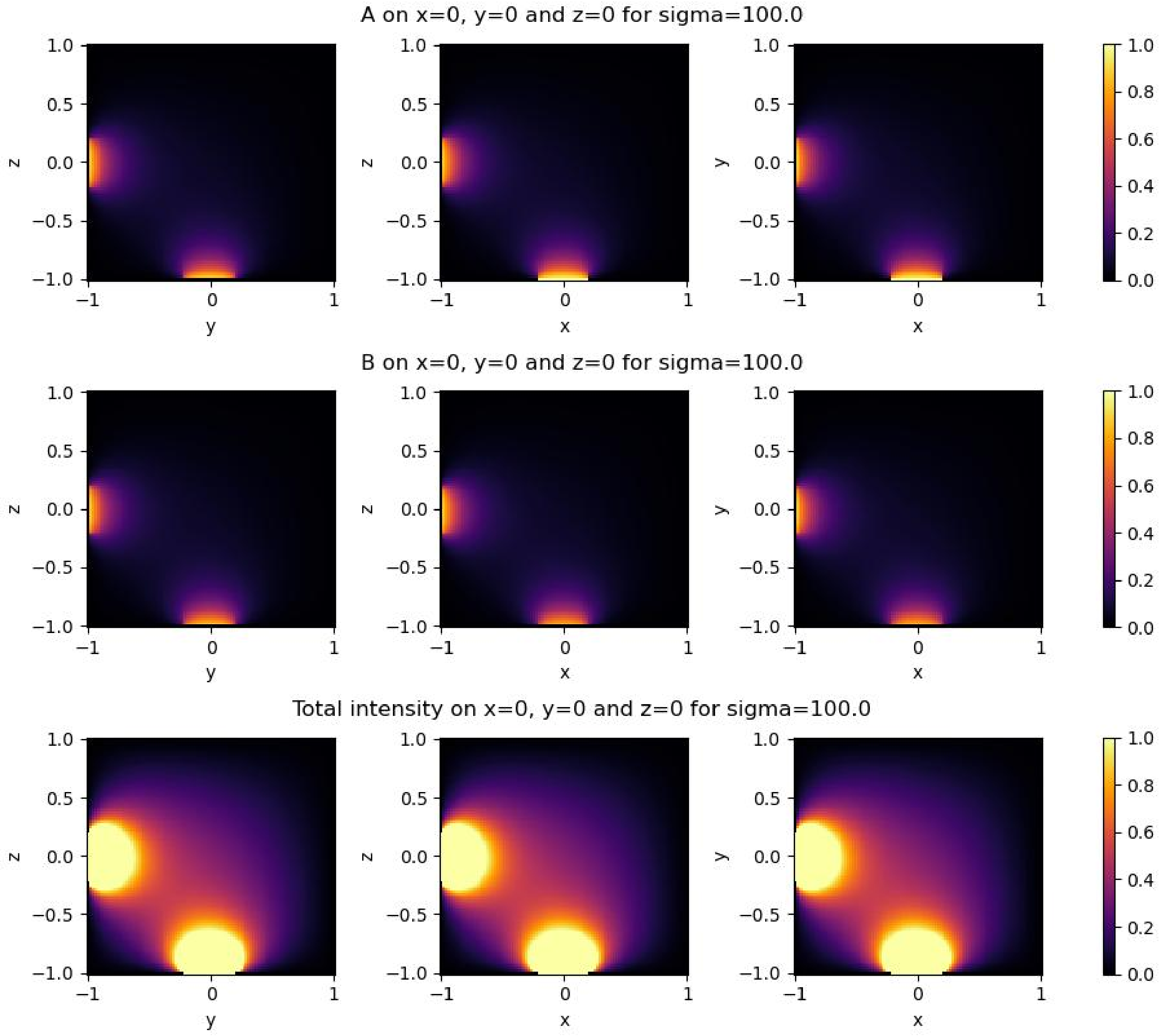


Figure 7: SGS solution for $\sigma = 100$



5 Conclusion

5.1 Revisiting the problem

This system of PDEs models Radiative Transfer: the physical phenomenon of energy transfer in the form of electromagnetic radiation (visible light, radio wave etc.). It considers six velocities in 3D space. A (or F_1) denotes the intensity of light transferring in the positive horizontal direction and B (or F_2) in the negative horizontal direction. Similarly, C, D, E and F denote intensity in their respective directions. The right-hand side is the scattering term: light is absorbed and emitted in all six directions. σ is the scattering coefficient. The boundary conditions are for light entering through a window from one side in each direction. With increase in σ the decay of intensity along the axes increases and light is scattered more.

5.2 Solving methods

Fixed-point (FP) iteration with relaxation and Symmetric Gauss-Seidel (SGS) with successive over-relaxation were employed to solve the system numerically for various values of σ. The choice of relaxation is not clear in either case and would depend on the value of σ. However, with some experimentation, it can drastically improve the convergence rate. SGS converges in fewer iterations but FP converges faster due to simpler iteration logic: (0.9s vs 0.2s per iteration). The algorithms were said to have converged when the residual, the numerical difference between the approximate left-hand side and the right-hand side (root sum squared), was less than the chosen threshold of 10^{-6}.

5.3 Observations

When $\sigma = 0$, A is constant along x and would be equal to the boundary values. When σ is very large, there's a steep drop in A at the very beginning and continues that way. In other words, the larger the σ, greater the decay. This can be observed in Figures 2, 4 and 5. It can also be observed that for large σ, area outside the axes also light up due to scattering.

Due to symmetry of the problem space and the equations, we have $a(x, y, z) = c(y, x, z) = e(z, y, x)$ and $b(x, y, z) = d(y, x, z) = f(z, y, x)$. So, the 2D plots plot only intensities A, B and the sum total. This could potentially be exploited for a synthetic SGS method.