WEEK 5

2020136006 정성원

Poisson equations

2차원 Poisson equation은 다음과 같이 주어진다.

$$\nabla^2 u(x,y) = f(x,y)\ for\ (x,y) \in \Omega$$

그리고 경계 $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$ 는 다음과 같이 주어진다.

$$u(x,y) = g(x,y)\ on\ \partial\Omega_D\ and\ \partial u/\partial n = h(x,y)\ on\ \partial\Omega_N$$

n은 경계에 대한 수직방향이며, $\partial\Omega_D$는 Dirichlet 경계를, $\partial\Omega_N$는 Neumann 경계를 의미한다.

1. (Iterative Poisson solver) 정사각 계산영역 $[0,1] \times [0,1]$ 이 주어졌고, 경계에서의 u=0이며, $f(x,y) = \sin(\pi x)\sin(\pi y)$로 주어졌을 때, 포아송 방정식을 계산하시오

(1) 반복계산을 사용하여 Poisson equation 을 균일 격자계에서 주변 5개 격자점을 사용하여 계산하시오. 1) Jacobi method, 2) Gauss-Seidel method, 3) Gauss-Seidel method with successive over-relaxation (SOR)

1-(1)을 풀기 위해 각각 Jacobi, Gauss-Seidel, SOR method 에 대해 알 필요가 있다.

Jacobi Method

다음 시점의 u 를 계산하기 위해 이전 반복의 u 값만을 사용해서 계산한다.

$$u_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}u_j^{(k)}\right)$$

Gauss-Seidel Method

Jacobi method 와 유사하지만 이미 계산된 최신값을 계산에 반영한다.

$$u_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}u_j^{(k+1)} - \sum_{j>i} a_{ij}u_j^{(k)}\right)$$

SOR method

Gauss-Seidel 에서 구한 값에 w 를 곱하여 수렴 속도를 인위적으로 빠르게 한다. w 를 이용하여 과거의 값과 현재의 값의 비율을 찾는다.

$$u_i^{(k+1)} = (1-\omega)u_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}u_j^{(k+1)} - \sum_{j>i} a_{ij}u_j^{(k)}\right)$$

Def 함수를 이용하여 $f = \sin(\pi x)\sin(\pi y)$를 정의한다.

```
def f(x, y):
    func = np.sin(np.pi * x) * np.sin(np.pi * y)
    return func
```

Jacobi method

Jacobi method 를 정의한다. 원래 u 행렬에 있는 값을 사용하는 것을 볼 수 있다.

```
# Jacobi method
def jacobi_method(u,h):
    u_new = np.copy(u)
    for i in range(1, len(u[0])-1):
        for j in range(1, len(u[0])-1):
            u_new[i, j] = (u[i+1, j] + u[i-1, j] + u[i, j+1] + u[i, j-1] - h**2 * f(X[i,j], Y[i,j])) / 4
    return u_new
```

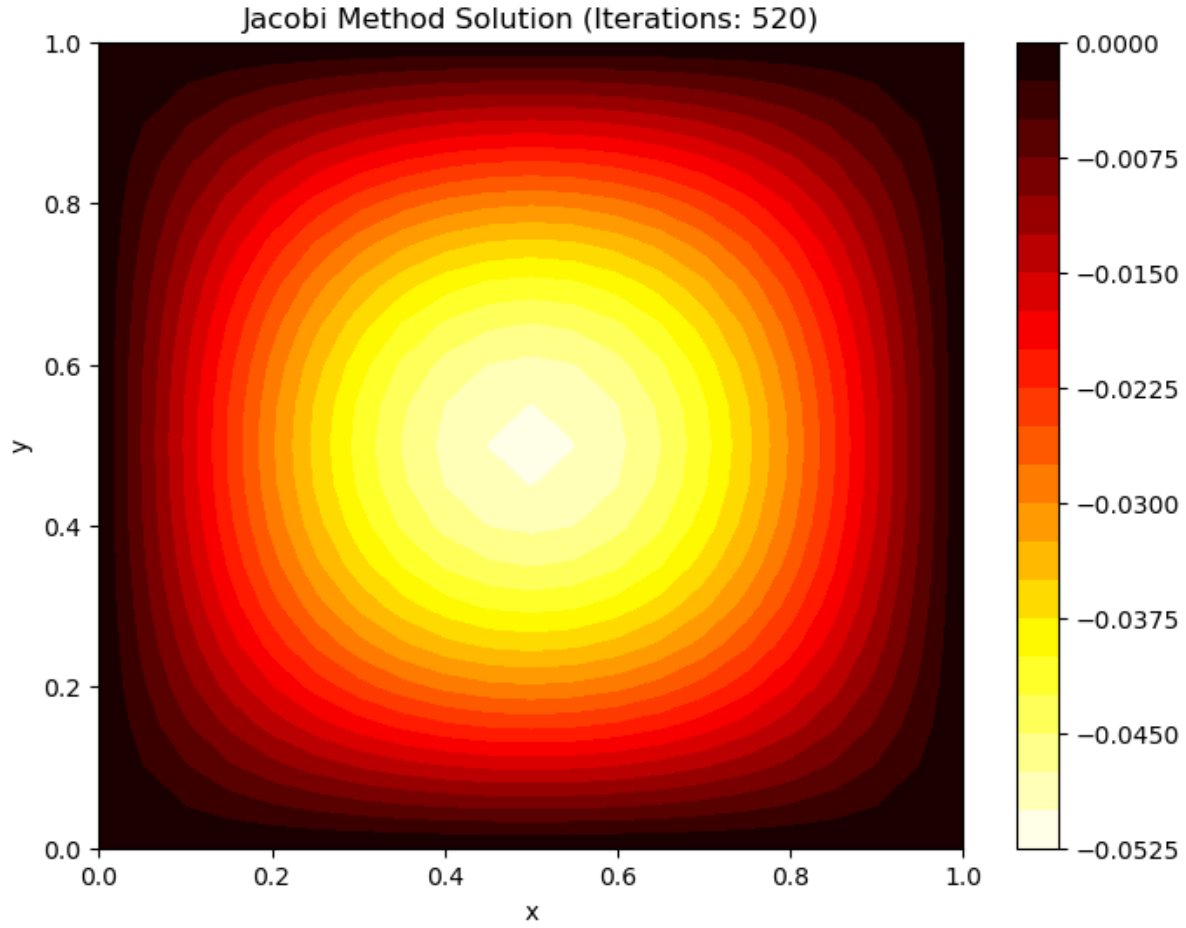
```
n = 21
x_list = np.linspace(0, 1, n) # x grid points
y_list = np.linspace(0, 1, n) # y grid points
X, Y = np.meshgrid(x_list, y_list) # create a meshgrid
h = x_list[1] - x_list[0] # grid spacing
u_0 = np.zeros((n, n)) # initial condition


u_list_jacobi = []
u_list_jacobi.append(u_0)
u_list_jacobi.append(jacobi_method(u_0, h))
iteration = 0

while np.linalg.norm(u_list_jacobi[-1] - u_list_jacobi[-2]) > 1e-5:
    iteration += 1
    u_jacobi = jacobi_method(u_list_jacobi[iteration], h)
    u_list_jacobi.append(u_jacobi)
    print(iteration)
```

초기값을 지정하고 Jacobi method 를 이용하여 나타낸 결과값을 u_list_jacobi 에 저장한다.

또한, while 을 이용하여 그전 값과의 차이가 10**-5 보다 작으면 jacobi method 를 정지하게 하였다.

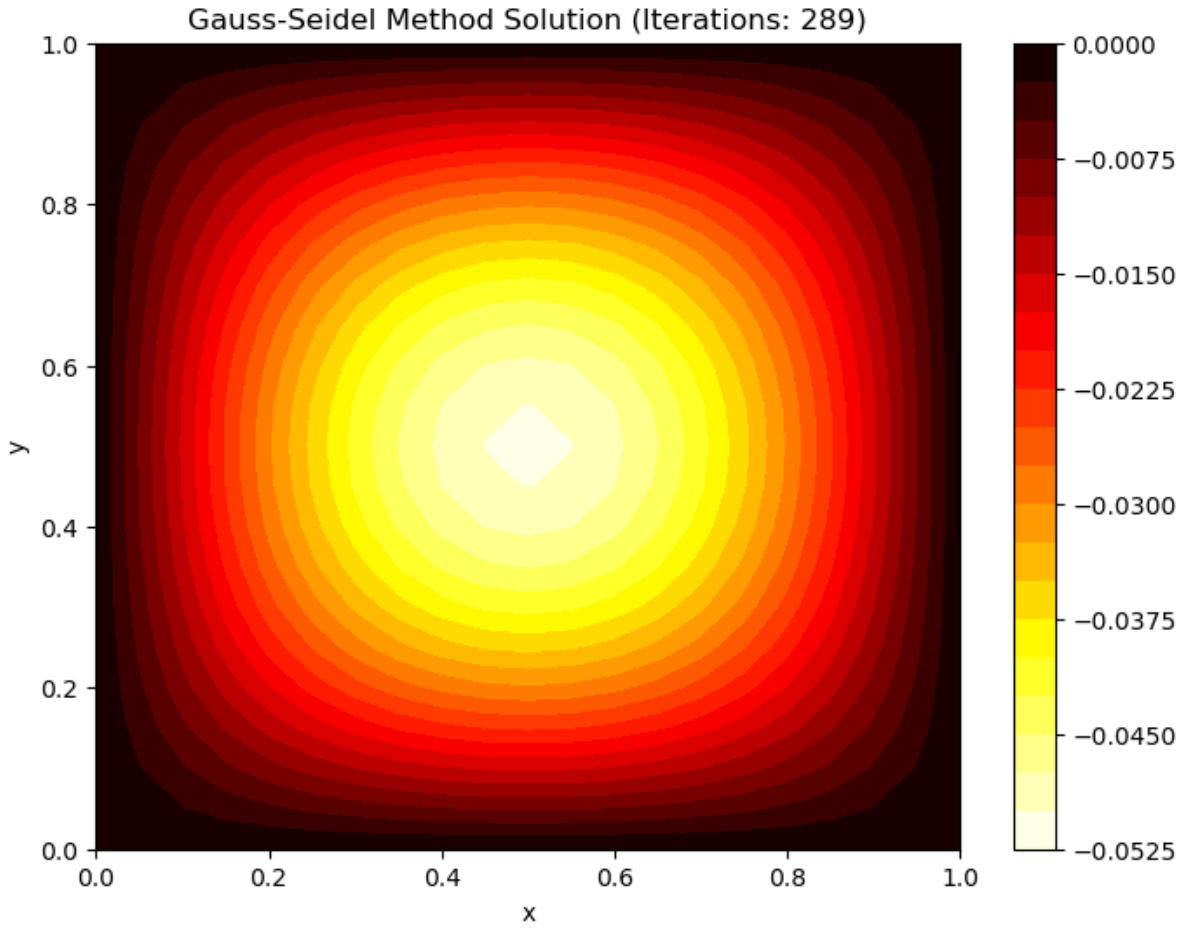


결과는 위와 같다.

Gauss-Seidel method

```
# Gauss-Seidel method
def gauss_method(u, h):
    u_new = np.copy(u)
    for i in range(1, len(u[0])-1):
        for j in range(1, len(u[0])-1):
            u_new[i, j] = (u_new[i+1, j] + u_new[i-1, j] + u_new[i, j+1] + u_new[i, j-1] - h**2 * f(X[i,j], Y[i,j])) / 4
    return u_new
```

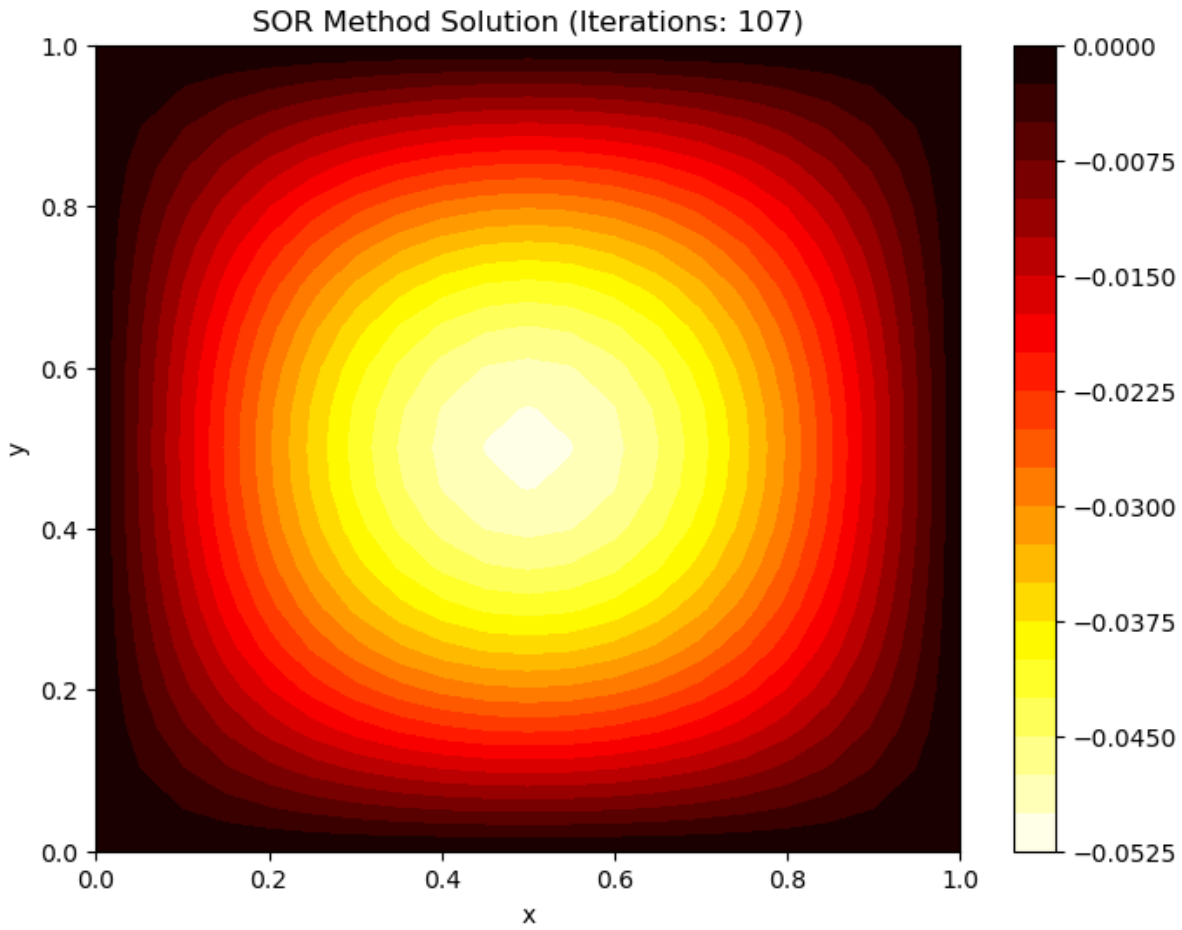
u 를 copy 하여 u_new 를 만들었기 때문에 과거 데이터가 저장되어 있고 for 문 안에 u_new 를 이용하여 계산을 진행하기 때문에 과거 데이터와 미래 데이터가 동시에 사용되는 것을 알 수 있다. 위과 같이 while 문을 사용하여 결과를 나타내면 아래와 같다.



SOR method

```python
# Gauss-Seidel method with successive over-relaxation (SOR)
def SOR_method(u, h, omega=1.5):
    u_new = np.copy(u)
    for i in range(1, len(u[0])-1):
        for j in range(1, len(u[0])-1):
            u_new[i, j] = (1 - omega) * u[i, j] + omega * (u_new[i+1, j] + u_new[i-1, j] + u_new[i, j+1] + u_new[i, j-1] - h**2 * f(X[i,j], Y[i,j])) / 4
    return u_new
```

위 함수를 보면 omega 를 사용하여 u 와 u_new 에 적절한 값을 곱하여 나타낸 것을 확인할 수 있다. 결과를 나타내면 아래와 같다.



(2) 각 반복계산 방법의 Performance를 보이시오. Ex) norm of residual, errors, computational time, etc

norm of residual 과 errors, interation 을 구하면 아래와 같다.

```
Jacobi Method
iteration: 520
Residual norm: 0.01573917632954167
error 0.00024395552097739048
```

```
Gauss-Seidel Method
iteration: 289
Residual norm: 0.0077607690410414305
error 0.0006547778704003776
```

```
SOR Method
iteration: 107
Residual norm: 0.002634609393248058
error 0.0009256998420178138
```

2. (Lineaerity) 다음의 포아송 방정식을 고려하여 문제를 푸시오.

$$\nabla^2 u(x,y) = f_1(x,y) + f_2(x,y)\ for\ (x,y) \in \Omega$$

그리고 경계 $\partial\Omega$에서 $u(x,y) = 0$ 를 만족하며, 정사각 계산영역 $[0,1] \times [0,1]$에서 진행하시오.

(1) Poisson equation의 해 u(x,y)를 SOR 방법을 사용하여 구하시오. 우항의 forcing 함수는 다음과 같이 주어진다.

$$f_1(x,y) = \sin(\pi x)\sin(\pi y)$$
$$f_2(x,y) = \exp(-100.0((x-0.5)^2 + (y-0.5)^2))$$

해당 문제에서 f1 과 f2 을 아래와 같이 정의한다.

```python
def f1(x,y):
    return np.sin(np.pi * x) * np.sin(np.pi * y)


def f2(x,y):
    return np.exp(-100*((x-0.5)**2 + (y-0.5)**2))
```

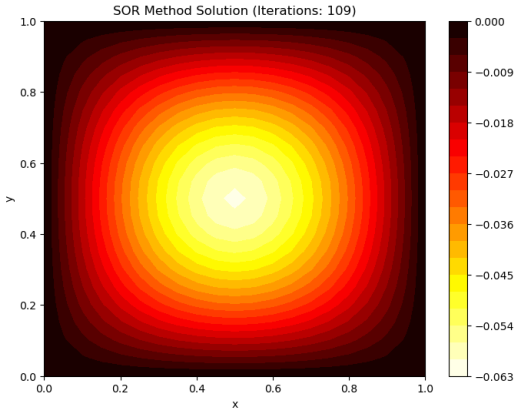
```python
# Gauss-Seidel method with successive over-relaxation (SOR)
def SOR_method2(u, h, omega=1.5):
    u_new = np.copy(u)
    for i in range(1, len(u[0])-1):
        for j in range(1, len(u[0])-1):
            u_new[i, j] = (1 - omega) * u[i, j] + omega * (u_new[i+1, j] + u_new[i-1, j] + u_new[i, j+1] + u_new[i, j-1] - h**2 * (f1(X[i,j], Y[i,j])+f2(X[i,j], Y[i,j])
    return u_new

# Gauss-Seidel method with successive over-relaxation (SOR)
def SOR_method2(u, h, omega=1.5):
    u_new = np.copy(u)
    for i in range(1, len(u[0])-1):
        for j in range(1, len(u[0])-1):
            u_new[i, j] = (1 - omega) * u[i, j] + omega * (u_new[i+1, j] + u_new[i-1, j] + u_new[i, j+1] + u_new[i, j-1] - h**2 * (f1(X[i,j], Y[i,j])+f2(X[i,j], Y[i,j])
    return u_new

def SOR_method3(u, h, omega=1.5):
    u_new = np.copy(u)
    for i in range(1, len(u[0])-1):
        for j in range(1, len(u[0])-1):
            u_new[i, j] = (1 - omega) * u[i, j] + omega * (u_new[i+1, j] + u_new[i-1, j] + u_new[i, j+1] + u_new[i, j-1] - h**2 * (f2(X[i,j], Y[i,j]))) / 4
    return u_new
```

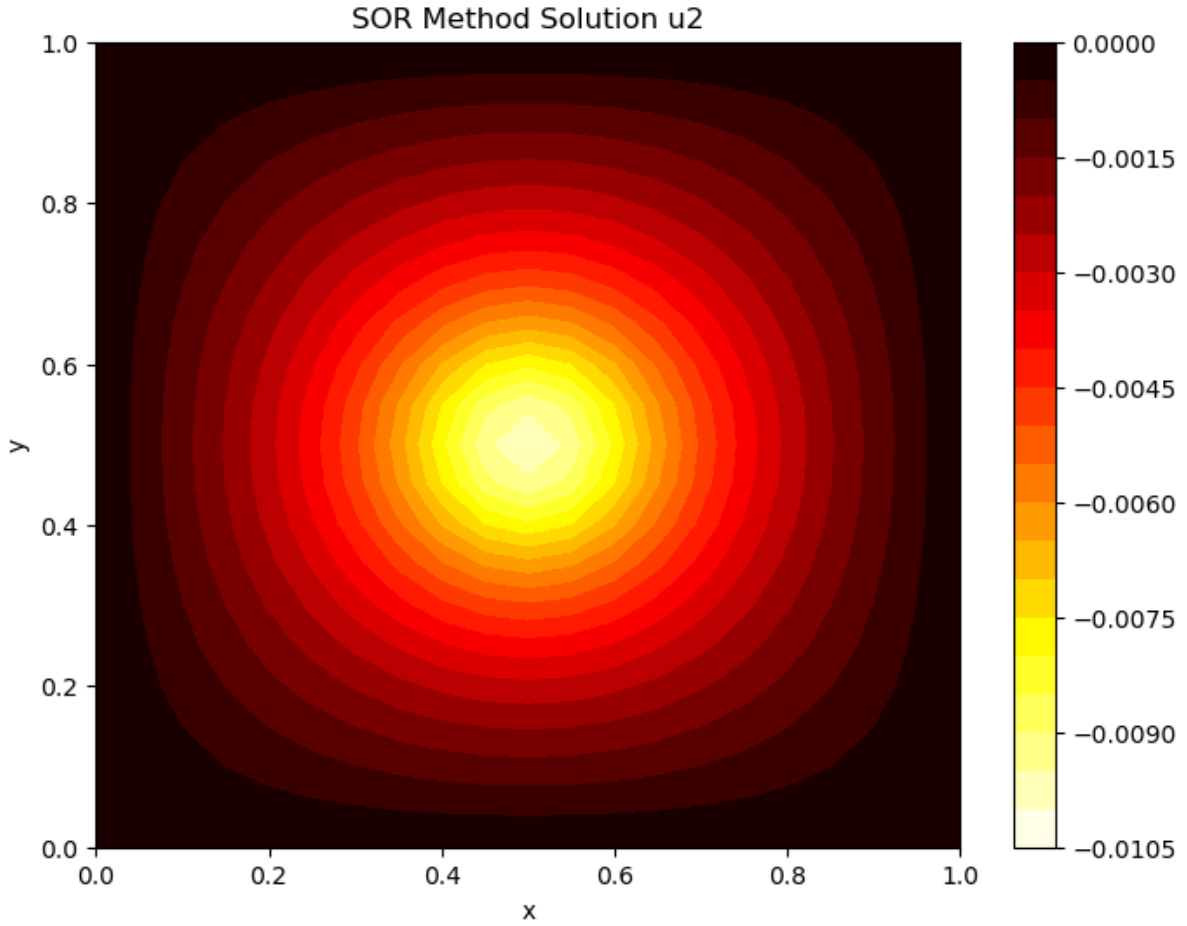
사용하는 f 값이 1 번 문제와 다르기 때문에 다시 정의해준다.

똑같이 while 문을 사용하여 문제를 풀고 결과를 나타내면 아래와 같다.



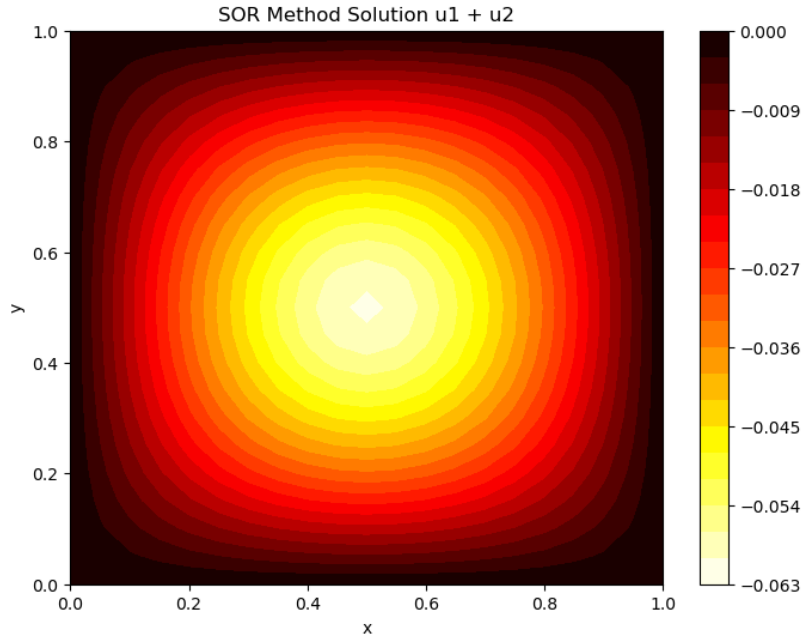
(2) 동일한 방법으로 f_2만을 고려한 포아송 방정식의 해 u_2를 구하시오

SOR_method3 함수에도 저장된 방법을 이용하여 u2 를 구하여 시각화 하면 다음과 같다.

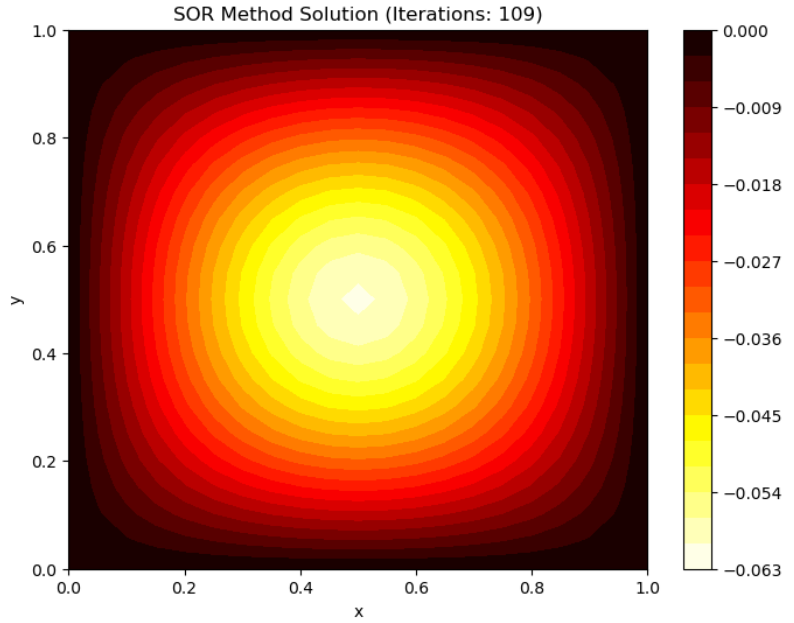


(3) 문제에서 구한 해 $u(x,y)$와 1. 문제에서 구한 $u_1(x,y)$ 그리고 2.-(2) 에서 구한 $u_2(x,y)$들의 해를 비교하고 이에 대한 생각을 서술하시오.

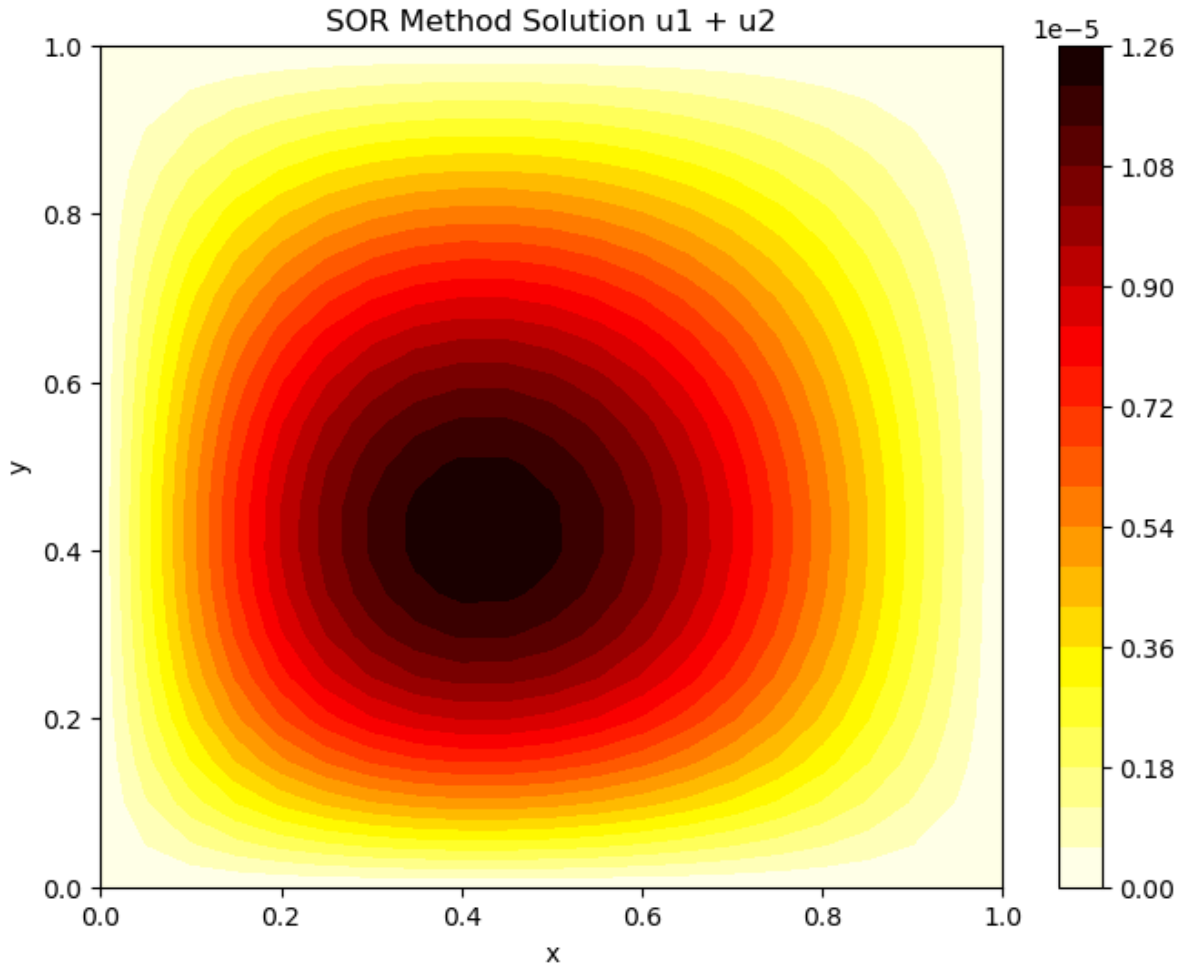
문제 1 에서 포아송 방정식이 선형이기 때문에 u1 과 u2 의 합이 u 라고 생각할 수 있다.



u1+u2 의 합을 그리면 아래와 같은 것을 확인할 수 있는데 문제 2 에서 구한 u 를 시각화하면 다음과 같다.



두 개의 데이터가 거의 동일하다는 것을 확인할 수 있다.



두개의 차이를 보면 위와 같은 것을 확인할 수 있는데 차이의 값을 10**-5 order 인 것을 확인할 수 있는데 이는 매우 작다는 것을 확인할 수 있다. 좌측으로 기울어지는 이유는 잘 모르겠다. 계산을 할 때 차수가 낮은 숫자에서 발생하는 오류가 등장하는 것인지 잘 모르겠다.