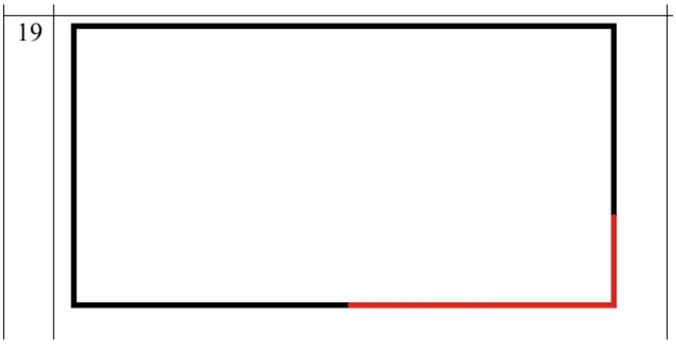
# Assignment

Shahnoza Nurubloeva 22B051055

## Problem




## Mathematical Model

Equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

BC & IC

$$u = \begin{cases} 1 & \text{if } x \in [L/2, L) \text{ and } y = 0 \text{ or } x = L \text{ and } y \in [0, L/2) \\ 0 & \text{otherwise} \end{cases}$$

## Numerical Approximation Formulas

Jacobi

$$u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n)$$

Gauss-Seidel

$$u_{i,j}^{n+1} = \frac{1}{4}(u_{i+1,j}^n + u_{i-1,j}^{n+1} + u_{i,j+1}^n + u_{i,j-1}^{n+1})$$

Relaxation Method

$$u_{i,j}^{n+1} = \frac{w}{4}(u_{i+1,j}^n + u_{i-1,j}^{n+1} + u_{i,j+1}^n + u_{i,j-1}^{n+1} - 4(1 - \frac{1}{w})u_{i,j}^n)$$

since $w \in (0,1)$ — under-relaxation — yeilds Gauss-Seidel Method

$w = 1/2 \in (1,2)$ was used

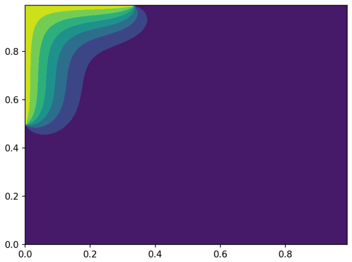
---

## Jacobi Method Code & Results

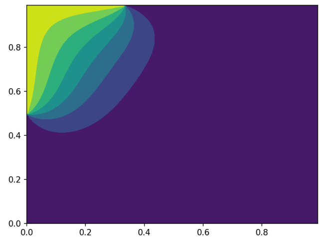
```
import matplotlib.pyplot as plt
n=100
dx=dy=1/n # nxn square plate that is heated
dt=0.001
itt=0

u=[]

# initial state of u
for i in range(n):
    u_row = []
    for j in range(n):
        if j==0 and (n/2<=i<=n-1) or (i==n-1 and 0<=j<=n/3):
            u_row.append(1)
        else:
            u_row.append(0)
    u.append(u_row)

xlist=[i*dx for i in range(n)]
ylist=[j*dy for j in range(n)]

while True:
    # iterations
    un = []
    error = 0

    # initial state of u for each iteration itt
    for i in range(n):
        u_row = []
        for j in range(n):
            if j==0 and (n/2<=i<=n-1) or (i==n-1 and 0<=j<=n/3):
```

```
            u_row.append(1)
        else:
            u_row.append(0)
    un.append(u_row)
for i in range(1, n-1):
    for j in range(1, n-1):
        un[i][j]=(1/4)*(u[i+1][j]+u[i-1][j]+u[i][j-1]+u[i][j+1])
        error = max(error, abs(un[i][j]-u[i][j]))
itt+=1

# to iterate repeatedly using prev data in order to find more precise u
u=un

if itt%300==0:
    print('still running')
    plt.contourf(xlist, ylist, u)
    plt.show()

if error<0.0001:
    break

print(itt)
plt.contourf(xlist, ylist, u)
plt.show()
```

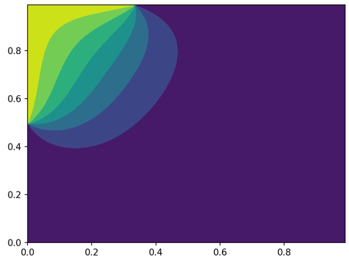
First




Intermediate




Last



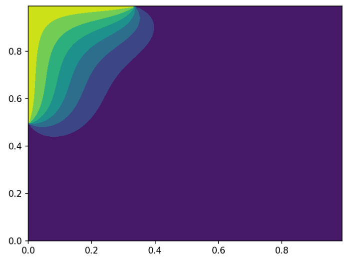

Number of iterations: 1733

## Gauss-Seidel Method Code & Results

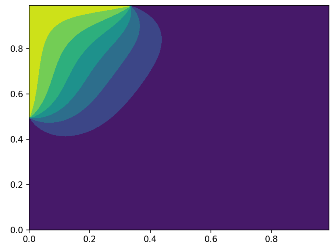
```
import matplotlib.pyplot as plt
n=100
dx=dy=1/n
dt=0.001
itt=0

u=[[]]

# initial state of u
for i in range(n):
    u_row = []
    for j in range(n):
        if j==0 and (n/2<=i<=n-1) or (i==n-1 and 0<=j<=n/3):
            u_row.append(1)
        else:
            u_row.append(0)
    u[0].append(u_row)

xlist=[i*dx for i in range(n)]
ylist=[j*dy for j in range(n)]

while True:
    error = 0
    u.append([])
    itt+=1
    for i in range(n):
        u_row = []
        for j in range(n):
            if j==0 and (n/2<=i<=n-1) or (i==n-1 and 0<=j<=n/3):
                u_row.append(1)
            else:
                u_row.append(0)
        u[itt].append(u_row)
    for i in range(1, n-1):
        for j in range(1, n-1):
            # here, u at steps j-1 and i-1 comes from n+1 (new/current iteration)
            # while u at steps j+1 and i+1 compes from n (old/prev)
            u[itt][i][j]=(1/4)*(u[itt-1][i+1][j]+u[itt][i-1][j]+u[itt][i][j-1]+u[itt-1][i][j+1])
```

```
            error = max(error, abs(u[itt][i][j]-u[itt-1][i][j]))

    if itt%300==0:
        print('still running')
        plt.contourf(xlist, ylist, u[-1])
        plt.show()

    if error<0.0001:
        break

print(itt)
plt.contourf(xlist, ylist, u[-1])
plt.show()
```

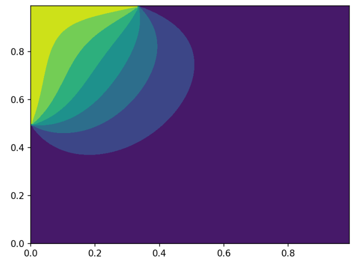
First




Intermediate




Last




Number of iterations: 1275

---

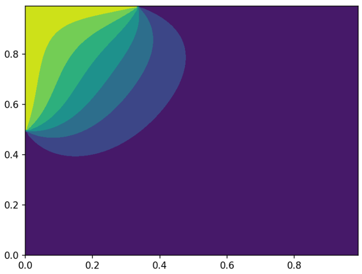
## Relaxation Method Code & Results

```
import matplotlib.pyplot as plt
n=100
dx=dy=1/n
dt=0.001
itt=0

# introducing relaxation coefficient
w=1.5

u=[[]]
```

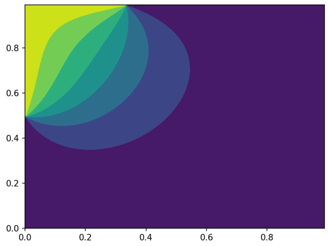
```
# initial state of u
for i in range(n):
    u_row = []
    for j in range(n):
        if j==0 and (n/2<=i<=n-1) or (i==n-1 and 0<=j<=n/3):
            u_row.append(1)
        else:
            u_row.append(0)
    u[0].append(u_row)

xlist=[i*dx for i in range(n)]
ylist=[j*dy for j in range(n)]

while True:
    error = 0
    u.append([])
    itt+=1
    for i in range(n):
        u_row = []
        for j in range(n):
            if j==0 and (n/2<=i<=n-1) or (i==n-1 and 0<=j<=n/3):
                u_row.append(1)
            else:
                u_row.append(0)
        u[itt].append(u_row)
    for i in range(1, n-1):
        for j in range(1, n-1):
            u[itt][i][j]=(w/4)*(u[itt-1][i+1][j]+u[itt][i-1][j]+u[itt][i][j-1]+u[itt-1][i][j+1]-4
            error = max(error, abs(u[itt][i][j]-u[itt-1][i][j]))

    if itt%300==0:
        print('still running')
        plt.contourf(xlist, ylist, u[-1])
        plt.show()

    if error<0.0001:
        break
```

```
print(itt)
plt.contourf(xlist, ylist, u[-1])
plt.show()
```

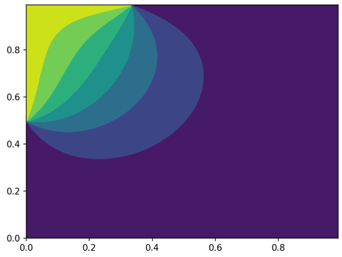
First



Intermediate



Last



Number of iterations: 737

## Conclusion

- Jacobi Method is easy to implement, but it often requires a large number of iterations. This indicates that its convergence rate is slower.
- Gauss-Seidel Method updated values immediately within each iteration, it converges faster then Jacobi method.
- Relaxation Method converges the fastes out of these three methods, if parameter $w$ is wisely chosen. Otherwise, it can lead to instability.