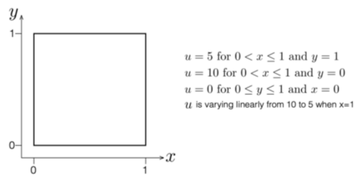
PDE python exercise 4

In this exercise we will apply an iterative method to solve a non-linear elliptic PDE. The problem we consider is a squared plate and in particular the distribution of a variable u for a given set of boundary conditions. This plate is illustrated in the figure below as well as its boundary conditions.



The plate is of size 1 in x and 1 in y. Dirichlet boundary conditions are applied to each edge of the plate. The distribution of our dependent variable u in the plate is governed by the following PDE which reads:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + u^2 = -1$$

We want to solve this problem using a centered finite difference for both partial derivatives and the Successive Over Relaxation method.

In this exercise, you need to :

On paper:

1. Discretize the PDE using the chosen finite differences
2. Formulate the iterative scheme using the Successive Over Relaxation method

Within the python notebook:

1. Discretize the independent variables x and y using a `numpy meshgrid`, both x and y should be discretized using 100 points (i.e `linspace(0.0,1.0,100)`)
2. Write a function which takes in the value of the SOR parameter (ω) as well as the desired tolerance for the iterative method and return the dependent variable u as well as the number of iterations needed to reach the given tolerance. The boundary conditions will be enforced within the function. The error commited by the iterative scheme should be computed using a normalized max norm of the increment δU
3. Plot the resulting dependent variable u using a filled contour plot when using a tolerance of 0.01 and $\omega = 1.5$.
4. Repeat the computation for a SOR parameter $\omega = 1.0$ and plot the difference between the new prediction and the one from point 5.
5. Repeat the computation for a tolerance of 0.001 and plot the difference between the new prediction and the one from point 5.
6. What can you say about the number of iterations required in points 5, 6 and 7?

Question 1:

We express our finite difference approximation of $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ as:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

and

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

where we have assumed that the discretization along x Δx and along y Δy are the same and defined as h.

When incorporating the finite difference approximations in our PDE, we can obtain the following numerical scheme driving our problem:

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} + h^2(u_{i,j}^2 + 1) = 0$$

Question 2

The SOR method, applied to a non-linear PDE is defined as:

$$u_{i,j}^{m+1} = u_{i,j}^m + \delta u_{i,j}$$

where m in an iteration counter.

The correction term $\delta u_{i,j}$ is expressed using Newton's method and reads:

$$\delta u_{i,j} = -\omega \frac{f_{i,j}}{\frac{\partial f}{\partial u_{i,j}}}$$

where ω is the relaxation parameter and $f_{i,j}$ is given by:

$$f_{i,j} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} + h^2(u_{i,j}^2 + 1)$$

Consequently the correction factor reads:

$$\delta u_{i,j} = -\omega \frac{u_{i+1,j}^m + u_{i-1,j}^{m+1} + u_{i,j+1}^m + u_{i,j-1}^{m+1} - 4u_{i,j}^m + h^2\left(\left(u_{i,j}^m\right)^2 + 1\right)}{-4 + 2 * h^2 u_{i,j}^m}$$

To determine which terms are computed at iteration m and $m+1$ it was assumed that the numerical scheme will be screened starting from the lower left of the plate towards the higher right corner.

Next we import the required python packages.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

The discretized independent variables x and y are obtained by a `numpy meshgrid`

In [ ]:
```
NX, NY = 100,100
X,Y = np.meshgrid(np.linspace(0.0,1.0,NX),np.linspace(0.0,1.0,NY))
```

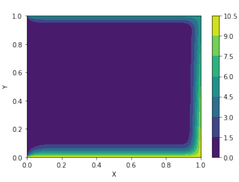
We create a function to compute the dependent variable u with the relaxation parameter ω and the tolerance of the iterative scheme as input. As output from this function we get the dependent variable u as well as the number of iterations carried out by the numerical scheme.

In [ ]:
```
def compute_u(omega,reltol):
#   Initialize the iteration and convergence
    iteration = 0
    rel_res   = 1.0
#   Initialize the dependent variable
    U = np.zeros([100,100])
#   Compute step size
    h = X[1,1]-X[1,2]
#   Boundary conditions
    U[0,:]  = 10.0
    U[-1,:] = 5.0
    U[:,-1] = np.linspace(10.0,5.0,NX)
#
    while (rel_res > reltol):
          du_max=0.0
          for j in range(1,NY-1):
              for i in range(1, NX - 1):
                   R  = (U[j,i-1]+U[j-1,i]+U[j,i+1]+U[j+1,i]-4.0*U[j,i]) + h**2*(U[j,i]**2.0+1.0)
                   df = 4.0-2.0*h**2*U[j,i]
                   dU = omega*R/df
                   U[j,i]+=dU
                   du_max=np.max([np.abs(dU),du_max])
          rel_res=du_max/np.max(np.abs(U))
          iteration+=1
    return U,iteration
```

We first carry out an analysis withe $\omega = 1.5$ and a tolerance of 0.01

In [ ]:
```
omega,reltol = 1.5, 0.01
U_1,iteration_1 = compute_u(omega,reltol)
plt.contourf(X,Y,U_1)
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar()
```

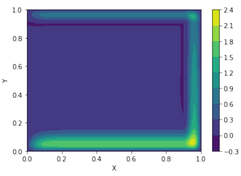
Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x13d7790a0>
```



We set carry out an analysis with $\omega = 1$ and plot the difference for u with the previous question. We can see that changing the relaxation parameter ω does have an impact on our solution.

In [ ]:
```
omega,reltol = 1.0, 0.01
U_2,iteration_2 = compute_u(omega,reltol)
plt.contourf(X,Y,U_1-U_2)
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar()
```

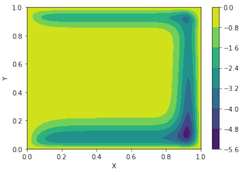
Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x13d82ad90>
```



We set carry out an analysis with $\omega = 1.5$ and a lower tolerance, and plot the difference for u with the first computation. We can see that decreasing the tolerance has an impact on our solution again.

In [ ]:
```
omega,reltol = 1.5, 0.001
U_3,iteration_3 = compute_u(omega,reltol)
plt.contourf(X,Y,U_1-U_3)
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x13d9037f0>
```



Finally, we print out the number of iteration required by the numerical scheme. It appears that lowering ω or decreasing the tolerance increases the number of iterations required by the scheme.

In [ ]:
```
print(iteration_1,iteration_2,iteration_3)
```

```
9 14 42
```