Computational Fluid Dynamics
Problem Set #4

MMAE 517 Spring 2020

Benjamin Thrun

Table of Contents

1 Functions Used **3**

2 Case 1: Re = 100 **9**
2.1 Convergence . . . 9
2.1.1 Convergence Code . . . 9
2.1.2 Convergence Results . . . 9
2.2 Plots . . . 10
2.2.1 Code for Plots . . . 10
2.2.2 Results for Plots . . . 11

3 Case 2: Re = 0 **11**
3.1 Convergence . . . 11
3.1.1 Convergence Code . . . 11
3.1.2 Convergence Results . . . 12
3.2 Plots . . . 12
3.2.1 Code for Plots . . . 12
3.2.2 Results for Plots . . . 13

4 Case 3: Grid Independence for Higher Reynolds Numbers **13**
4.1 Varying Reynolds Number . . . 14
4.1.1 Code . . . 14
4.1.2 Plots and Results . . . 16
4.2 Grid Independence for Re = 110 . . . 16
4.2.1 Code . . . 16
4.2.2 Results . . . 19
4.3 Plots of Grid Independent Solution . . . 21

5 Derivations **21**

1 Functions Used

[ ]:
```
########################### PROBLEM SET 4 ####################################
# Introduction

# In the vorticity-streamfunction formulation, the steady, incompressible flow
# in the cavity is governed by the vorticity-transport equation

# u(/x) + v(/y) = (1/Re)((2/x2)+(2/y2)), (1)

# and a Poisson equation for the streamfunction

# 2/x2 + 2/y2 = . (2)

# The boundary condition for (2) is  = 0 on all boundaries.

# Using the ADI Poisson solver you developed in Problem Set #2 and your
# ADI algorithm for the scalar transport equation developed in Problem Set
# 3,

# develop an iterative algorithm to solve the coupled equations (1) and (2).

# 1. Use Thom's method to obtain Dirichlet boundary conditions for the vorticity
# transport equation,

# 2. use central difference approximations to evaluate the
# definitions of the streamfunction, u = /y, v = /x, to obtain updated
# values of the velocities u(x, y) and v(x, y) for use in this equation. Obtain
# solutions for Re = 0 and Re = 100 on a 101 × 101 grid. If possible plot the
# streamlines and the iso-vorticity lines. Note that it may be necessary to use
# different acceleration parameters for each equation.
# Obtain what you regard to be grid-independent solutions for higher Reynolds
# numbers
```

[2]:
```
##################### Scalar Transport Solver ################################
def transport(l, I, alpha, u, v, phi, sigma):

# ideal acceleration sigmas 0.0174 from trial and error for alpha= 0.01
# ideal acceleration sigmas 0.111 from trial and error for alpha= 0.1

# Import libraries
    import numpy as np
    from thomas_algorithm import thomas

    N = I + 1 # Number of gridpoints
    delta = l/I # Length of the subinterval - deltax or deltay (equal)
```

```
    # Grid:
    # Set-up a, b, c, for the Thomas algorithm
    a = np.zeros(N)
    b = np.zeros(N)
    c = np.zeros(N)
    d = np.zeros(N)

############################# MAIN LOOP #######################################
######################### HORIZONTAL SOLVES ###################################

    for i in range(1, N-1):

        for j in range(0, N):

# a b c for Thomas
            if v[i,j] > 0.:
                a[j] = alpha + v[i,j]*delta
                b[j] = -(2.*alpha + v[i,j]*delta + sigma)
                c[j] = alpha

            elif v[i,j] < 0.:
                a[j] = alpha
                b[j] = -2.*alpha + v[i,j]*delta - sigma
                c[j] = alpha - v[i,j]*delta

            else: # v[i,j] == 0
                a[j] = alpha
                b[j] = -2.*alpha - sigma
                c[j] = alpha

# d for dirchlet BC
        for j in range(0, N):

            if u[i,j] > 0.:
                d[j] = (-alpha - u[i,j]*delta)*phi[i-1,j] + (2.*alpha + 
 ↪u[i,j]*delta -sigma)*phi[i,j] + -alpha*phi[i+1,j]

            elif u[i,j] < 0.:
                d[j] = (-alpha)*phi[i-1,j] + (2.*alpha - u[i,j]*delta 
 ↪-sigma)*phi[i,j] + (-alpha + u[i,j]*delta)*phi[i+1,j]

            else: # u[i,j] == 0
                d[j] = (-alpha)*phi[i-1,j] + (2.*alpha - sigma)*phi[i,j] + 
 ↪-alpha*phi[i+1,j]

# BC's for Thomas
        left = [1., 0., phi[i, 0]]
```

```
        right = [1., 0., phi[i, N-1]]

# Solve i's column:
        phi[i, :] = thomas(N, delta, a, b, c, d, left, right)


########################### VERTICAL SOLVES ###################################

    for j in range(1, N-1):

        for i in range(0, N):

# a b c for Thomas
            if u[i,j] > 0.:
                a[i] = alpha + u[i,j]*delta
                b[i] = -(2.*alpha + u[i,j]*delta + sigma)
                c[i] = alpha

            elif u[i,j] < 0.:
                a[i] = alpha
                b[i] = -2.*alpha + u[i,j]*delta - sigma
                c[i] = alpha - u[i,j]*delta

            else: # u[i,j] == 0
                a[i] = alpha
                b[i] = -2.*alpha - sigma
                c[i] = alpha

# d coefficient for dirchlet BC
        for i in range(0, N):

            if v[i,j] > 0.:
                d[i] = (-alpha - v[i,j]*delta)*phi[i,j-1] + (2.*alpha +␣
↪v[i,j]*delta -sigma)*phi[i,j] + -alpha*phi[i,j+1]

            elif v[i,j] < 0.:
                d[i] = (-alpha)*phi[i,j-1] + (2.*alpha - v[i,j]*delta␣
↪-sigma)*phi[i,j] + (-alpha + v[i,j]*delta)*phi[i,j+1]

            else: # v[i,j] == 0
                d[i] = (-alpha)*phi[i,j-1] + (2.*alpha - sigma)*phi[i,j] +␣
↪-alpha*phi[i,j+1]

# BC's for Thomas
        left = [1., 0., phi[0, j]]
        right = [1., 0., phi[N-1, j]]
```

```
# Solve j's row:
        phi[:, j] = thomas(N, delta, a, b, c, d, left, right)

# Compare the new solution to the previous iteration's solution

    return(phi)
```

[3]:
```
##################### ADI Poisson Solver ######################################

def poisson_solver(l, I, f, u, sigma):
    import numpy as np
    from thomas_algorithm import thomas

    N = I + 1 # Number of gridpoints
    delta = l/I # Length of the subinterval


# Horizontal solves:

    for j in range(1, N-1):
        a = np.full((N), 1.) # Lower diagonal
        b = np.full(N, -2. -sigma) # Main diagonal
        c = np.full((N), 1.) # Upper diagonal

        d = delta**2 * f[:, j] - u[:, (j + 1)] + (2. - sigma) * u[:, j] - u[:,(j␣
 ↪- 1)]
        left = [1., 0., u[0, j]]
        right = [1., 0., u[N-1, j]]

        u[:, j] = thomas(N, delta, a, b, c, d, left, right)

# Vertical Solves
    for i in range(1, N-1):
        a = np.full((N), 1.) # Lower diagonal
        b = np.full(N, -2. -sigma) # Main diagonal
        c = np.full((N), 1.) # Upper diagonal

        d = delta**2. * f[i, :] - (u[(i+1), :] - (2.-sigma) * u[i, :] + u[(i-1),␣
 ↪:]) # Interior
        left = [1., 0., u[i, 0]]
        right = [1., 0., u[i, N-1]]
        u[i, :] = thomas(N, delta, a, b, c, d, left, right)


    u = u - u[N//2, N//2] # recenter

    #u = u - u[N//2, N//2]
```

```
    return(u)
```

[9]:
```
############################## Solver ########################################
# Create solver that uses both vorticity transoport and streamfunction solvers

def solver(I, Re, tol, sigma_Vort, sigma_Stream):
    import numpy as np
    import copy
####################### Initialize Terms #####################################
    l = 1. # domain length
    N = I + 1 # number of grids
    delta = l / I # change between grids
    alpha = 1./Re # relationshiop found for alpha by derivations

# Initialize arrays for vorticity and streamfunction
    omega = np.zeros((N, N)) # vorticity
    psi = np.zeros((N, N)) # stream function

# Initialize the arrays of velocities
    u = np.zeros((N, N))
    v = np.zeros((N, N))

# convergence criterion and iteration counter
    iteration = 0 # iteration counter
    e_Vort = 1. # convergence criterion for vorticity
    e_Stream = 1. # convergence criterion for streamfunction

############################ MAIN LOOP #######################################

    while (e_Vort > tol) or (e_Stream > tol):

######################## TRANSPORT SOLVER ####################################

# Store past voricity matrix to compare to

        prev_omega = copy.copy(omega)

# Compute the velocity field from the streamfunction

        for i in range(1, N-1):

            for j in range(1, N-1):

                u[i, j] = (psi[i, j+1] - psi[i, j-1]) / (2.* delta)

                v[i, j] = (psi[i-1, j] - psi[i+1, j]) / (2.* delta)
```

```
# BC's for vorticity

        omega[:, 0] = (2./ delta**2) * (psi[:, 0] - psi[:, 1]) # Lower BC

        omega[:, N-1] = (2./ delta**2) * (psi[:, N-1] - psi[:, N-2] - delta) #␣
 ↪Upper BC

        omega[0, :] = (2./ delta**2) * (psi[0, :] - psi[1, :]) # Left BC

        omega[N-1, :] = (2./ delta**2) * (psi[N-1, :] - psi[N-2, :]) # Right BC


# Use transport function to calculate vortivity

        omega = transport(l, I, alpha, u, v, omega, sigma_Vort)

# Convergence Check For Vorticity

        diff_omega = omega - prev_omega # difference between current and past␣
 ↪iteration of vorticity
        enumer_omega = abs(max(diff_omega.min(), diff_omega.max(), key=abs))
        edenom_omega = abs(max(omega.min(), omega.max(), key=abs))
        e_Vort = enumer_omega / edenom_omega # new convergence error for␣
 ↪vorticity

####################### POISSON SOLVER #########################################

# store past iteration of streamfunction

        prev_psi = copy.copy(psi)

# Use poisson solver for streamfunction

        psi = poisson_solver(l, I, -omega, psi, sigma_Stream)

# Convergence check for the streamfunction:

        diff_psi = psi - prev_psi
        enumer_psi = abs(max(diff_psi.min(), diff_psi.max(), key=abs))
        edenom_psi = abs(max(psi.min(), psi.max(), key=abs))
        e_Stream = enumer_psi / edenom_psi

# Add to iteration

        iteration = iteration + 1
```

```
        print('iteration =', iteration) # iteration counter
        print('e_Vort = ', e_Vort) # convergence criterion of vorticity
        print('e_Stream = ', e_Stream) # convergence criterion of streamfunction
        print(N)
# Calculate the average omega and psi, take average velocity of vertical line in␣
 ↪the center

    omega_avg = np.sum(omega[:, :])/N # omega average
    psi_avg = np.sum(psi[:, :])/N # psi average

    return(omega, omega_avg, psi, psi_avg, I, e_Vort, e_Stream, iteration,␣
 ↪delta, l)
```

2 Case 1: Re = 100

2.1 Convergence

2.1.1 Convergence Code

[7]:
```
############################# CASE 1 ##########################################
#Re = 100
import numpy as np

# run function #I, Re, tol, sigma_Vort, sigma_ Stream #0.034, 1.489 # 1048␣
 ↪iterations
test1 = solver(101, 100, 1e-5,  0.034, 1.489)
omega, omega_avg, psi, psi_avg, I, e_Vort, e_Stream, iteration, delta, l = test1

############################# PRINT ###########################################

print("\n\nFor Re = 100, and I =", I, " grid divisions. The solution has␣
 ↪converged in", iteration, " iterations")

print("The convergence criterion of e_omega = ", e_Vort, " for vorticity, and␣
 ↪e_psi = ", e_Stream, " for streamfunction was achieved ")
```

2.1.2 Convergence Results

```
For Re = 100, and I = 101  grid divisions. The solution has converged in 1048
iterations

The convergence criterion of e_omega =  1.3782732995474474e-07  for vorticity,
```

```
and e_psi =  9.921819235638002e-06  for streamfunction was achieved
```

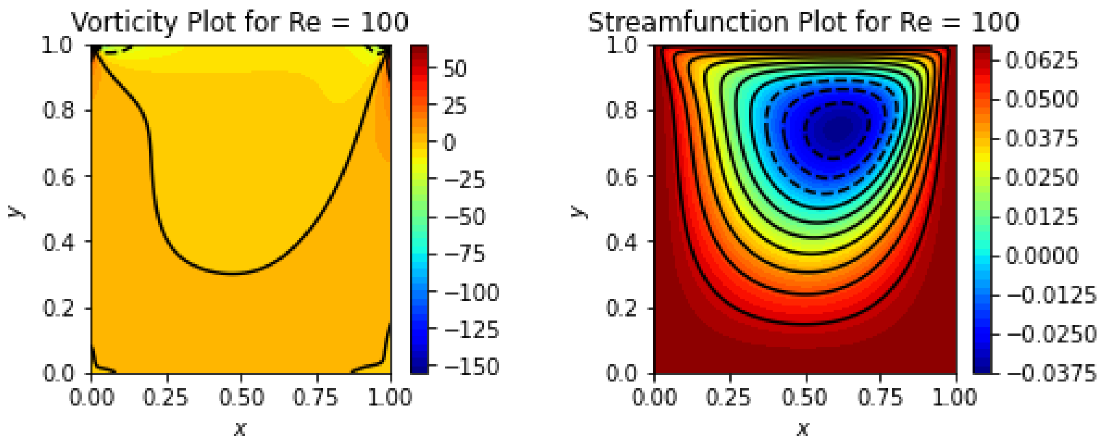
2.2 Plots

2.2.1 Code for Plots

[8]:
```
######################### PLOT RESULTS CASE 1 ##############################
import matplotlib.pyplot as plt
import scipy.constants

N = I+1
l = 1.
delta = l / I
x = np.linspace(0, l, N)
y = np.linspace(0, l, N)
X, Y = np.meshgrid(x, y)

fsy=6
fig = plt.figure(figsize = (1.25*fsy, fsy)) # Set the figure size to square
left = 0.1 # the left side of the subplots of the figure
right = 0.9 # the right side of the subplots of the figure
bottom = 0.15 # the bottom of the subplots of the figure
top = 0.5 # the top of the subplots of the figure
wspace = 0.5 # the amount of width reserved for space between subplots,
# expressed as a fraction of the average axis width
hspace = .25 # the amount of height reserved for space between subplots,
# expressed as a fraction of the average axis height
plt.subplots_adjust(left, bottom, right, top, wspace, hspace)


# plot vorticity
plt.subplot(1,2,1)
plt.contourf(Y, X, omega, 50, cmap = 'jet')
plt.colorbar()
plt.contour(Y, X, omega, 10, colors = 'k')
plt.title('Vorticity Plot for Re = 100')
plt.xlabel("$x$")
plt.ylabel("$y$")


# plot streamfunction
plt.subplot(1,2,2)
plt.contourf(Y, X, psi, 50, cmap = 'jet')
plt.colorbar()
plt.contour(Y, X, psi, 10, colors = 'k')
plt.title('Streamfunction Plot for Re = 100')
plt.xlabel("$x$")
plt.ylabel("$y$")
```

```
plt.show()
```

2.2.2 Results for Plots



3 Case 2: Re = 0

An important note prior to beginning is the Re = 0 would lead to an undefined solution due to a division by zero. Therefore, Re = 0.1 will be used instead to approximate the case for Re = 0.

3.1 Convergence

3.1.1 Convergence Code

[9]:
```
############################## CASE 2 ##########################################
# Re ~= 0, Re can't be zero since it is very low so need use Re = 0.1
# to compensate for a small reynolds numbers, acceleration paramaeters must
# be adjusted
# sigma_Vort = 3.1
# sigma_Stream = 1.0021
import numpy as np


# run function #(I, Re, tol, sigma_Vort, sigma_ Stream) #0.2,10,1.31
test2 = solver(101,  0.1, 1e-5, 20, 1.31)
omega, omega_avg, psi, psi_avg, I, e_Vort, e_Stream, iteration, dx, l = test2

############################## PRINT ###########################################
```

```
print("\n\nFor Re = 0, and I =", I, " grid divisions. The solution has converged␣
↪in", iteration, " iterations")

print("The convergence criterion of e_omega = ", e_Vort, " for vorticity, and␣
↪e_psi = ", e_Stream, " for streamfunction was achieved ")
```

3.1.2 Convergence Results

```
For Re = 0, and I = 101  grid divisions. The solution has converged in 672
iterations

The convergence criterion of e_omega =  8.353445290779422e-08  for vorticity,
and e_psi =  9.883159456997435e-06  for streamfunction was achieved
```

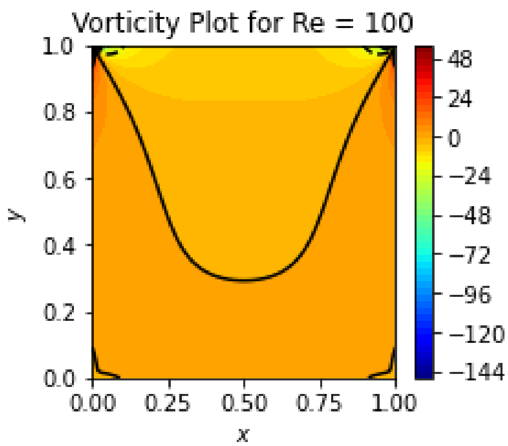
3.2 Plots

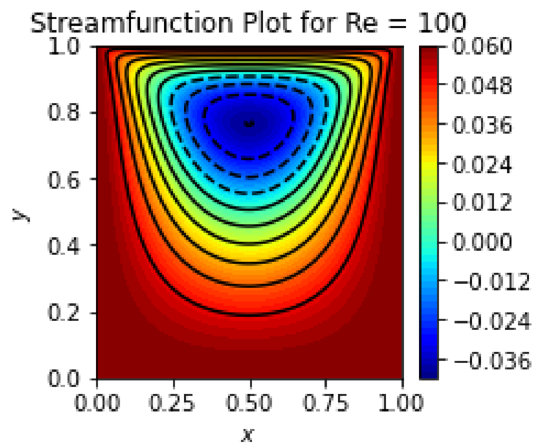
3.2.1 Code for Plots

[10]:
```
######################### PLOT RESULTS CASE 2 #########################
import matplotlib.pyplot as plt
import scipy.constants

N = I+1
l = 1.
delta = l / I
x = np.linspace(0, l, N)
y = np.linspace(0, l, N)
X, Y = np.meshgrid(x, y)

fsy=6
fig = plt.figure(figsize = (1.25*fsy, fsy)) # Set the figure size to square
left = 0.1 # the left side of the subplots of the figure
right = 0.9 # the right side of the subplots of the figure
bottom = 0.15 # the bottom of the subplots of the figure
top = 0.5 # the top of the subplots of the figure
wspace = 0.5 # the amount of width reserved for space between subplots,
# expressed as a fraction of the average axis width
hspace = .25 # the amount of height reserved for space between subplots,
# expressed as a fraction of the average axis height
plt.subplots_adjust(left, bottom, right, top, wspace, hspace)


# plot vorticity
plt.subplot(1,2,1)
plt.contourf(Y, X, omega, 50, cmap = 'jet')
```

```
plt.colorbar()
plt.contour(Y, X, omega, 10, colors = 'k')
plt.title('Vorticity Plot for Re = 100')
plt.xlabel("$x$")
plt.ylabel("$y$")


# plot streamfunction
plt.subplot(1,2,2)
plt.contourf(Y, X, psi, 50, cmap = 'jet')
plt.colorbar()
plt.contour(Y, X, psi, 10, colors = 'k')
plt.title('Streamfunction Plot for Re = 100')
plt.xlabel("$x$")
plt.ylabel("$y$")

plt.show()
```

3.2.2 Results for Plots





4 Case 3: Grid Independence for Higher Reynolds Numbers

Grid independence of higher Reynolds numbers was found by first looking at how vorticity and streamfunction changes with respect to a varying Reynolds number. Keeping grid size constant, Reynolds number was change from 10 to 180 with a step of 10 each time.

4.1 Varying Reynolds Number

4.1.1 Code

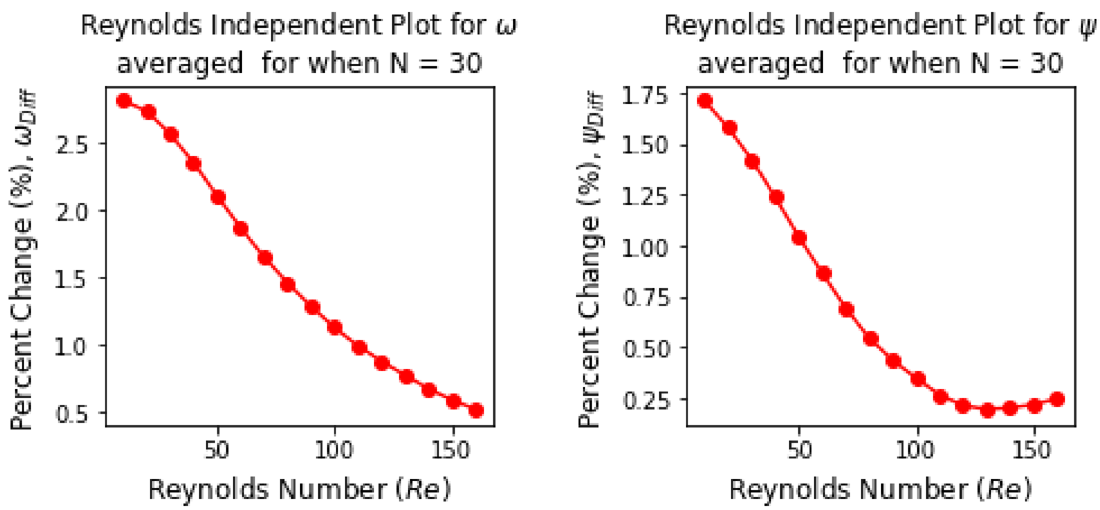
[11]:
```
############################## CASE 3 REYNOLDS SELECTION ########################
# Grid indpendence for high reynolds numbers, solve by increasing reynolds
# number increasing and investigationg how it changes.
# Use 30 grid intervals to test changing reynolds number
# Essentially find Reynolds number independence to find grid independence

#################### Determine Reynolds Independence ########################
# Grid independence will be determined by looking at the change in the average
# phi, both cases of alpha will be observed

import numpy as np
import matplotlib.pyplot as plt

# Changing Reynolds Number
k = np.arange(10,180,10)
test_5 = []
for Re in k:

                 # (N, Re, tolerance, sigma, sigma)
    test5 = solver(30, Re, 1e-5, 0.3, 1.49)
    test_5.append(test5)
```

[23]:
```
########################### Case 3 Plot ####################################

# Understand and interpret data to check reynolds number
# Take average of vorticity along the N/2 vertical line to have a
# good understanding of the data

omega_diff1 = np.zeros(len(k)-1) # vorticity
psi_diff1 = np.zeros(len(k)-1) # streamfunction


for r in range(0,len(k)-1):

    omega1, omega_avg1, psi1, psi_avg1, I, e_Vort, e_Stream, iteration, dx, l  =␣
 ↪test_5[r]
    omega2, omega_avg2, psi2, psi_avg2, I, e_Vort, e_Stream, iteration, dx, l  =␣
 ↪test_5[r+1]

    omega_diff1[r] = ((abs(omega_avg1 - omega_avg2)) / abs(omega_avg1 )) * 100
    psi_diff1[r] = ((abs(psi_avg1 - psi_avg2)) / psi_avg1 ) * 100
```

```python
# Plots
##############################################################################
# Convergence Plot (Change in Vorticity vs. Reynolds)
fsy=6
fig = plt.figure(figsize = (1.25*fsy, fsy)) # Set the figure size to square
left = 0.1 # the left side of the subplots of the figure
right = 0.9 # the right side of the subplots of the figure
bottom = 0.15 # the bottom of the subplots of the figure
top = 0.5 # the top of the subplots of the figure
wspace = 0.5 # the amount of width reserved for space between subplots,
# expressed as a fraction of the average axis width
hspace = .25 # the amount of height reserved for space between subplots,
# expressed as a fraction of the average axis height
plt.subplots_adjust(left, bottom, right, top, wspace, hspace)

plt.subplot(1,2,1)
plt.plot(k[0:-1], omega_diff1, color = 'r', marker = 'o')
plt.xlabel('Reynolds Number $(Re)$', fontsize =12)
plt.ylabel('Percent Change (%), $\omega_{Diff}$ ', fontsize =12)
plt.title('Reynolds Independent Plot for $\omega$ \naveraged  for when N = 30')

# Convergence Plot (Change in Streamfunction vs. Reynolds)
plt.subplot(1,2,2)
plt.plot(k[0:-1], psi_diff1, color = 'r', marker = 'o')
plt.xlabel('Reynolds Number $(Re)$', fontsize =12)
plt.ylabel('Percent Change (%), $\psi_{Diff}$ ', fontsize =12)
plt.title('Reynolds Independent Plot for $\psi$ \naveraged  for when N = 30')
plt.show()

##############################################################################
# now check convergence based on change in omega For vorticity
for l in range(0, len(omega_diff1)):

    if abs(omega_diff1[l]) < 1.0:
        break


Num_reynol = k[l]
print(' For a Reynolds independent solution with less than 1.0% change in vorticity, requrires, '
      'a Reynolds Number =', Num_reynol, 'for a reynolds independent solution.')

# now check convergence based on change in omega For streamfunction
for l in range(0, len(psi_diff1)):
```

```
    if abs(psi_diff1[l]) < 1.0:
        break


Num_reynol = k[l]
print(' For a Reynolds independent solution with less than 1.0% change in␣
 ↪streamfunction, requrires, '
      'a Reynolds Number =', Num_reynol, 'for a reynolds independent solution.')
```

4.1.2 Plots and Results




```
For a Reynolds independent solution with less than 1.0% change in vorticity,
requrires, a Reynolds Number = 110 for a reynolds independent solution.

For a Reynolds independent solution with less than 1.0% change in
streamfunction, requrires, a Reynolds Number = 60 for a reynolds independent
solution.
```

Since the Reynolds number independence was difference for vorticity and the streamfunciton, Re = 110 was chosen to look at for grid independence since it is the higher Reynolds to chose.

4.2 Grid Independence for Re = 110

4.2.1 Code

[11]:
```
############################# CASE 3 GRID INDEPENDENCE ########################
# Use Re = 110 from reynolds independence to evaluate grid independence

##################### Determine GRID INDEPENDENCE #############################
# Grid independence will be determined by looking at the change in the average
```

```
# phi, both cases of alpha will be observed
# use a step by 5 from 10 to 180 grids

import numpy as np
import matplotlib.pyplot as plt

# Changing Reynolds Number
k = np.arange(10,130,5)
test_5 = []
for N in k:

                  # (N, Re, tolerance, sigma, sigma)
    test5 = solver(N, 110, 1e-5, 0.034, 1.49)
    test_5.append(test5)
```

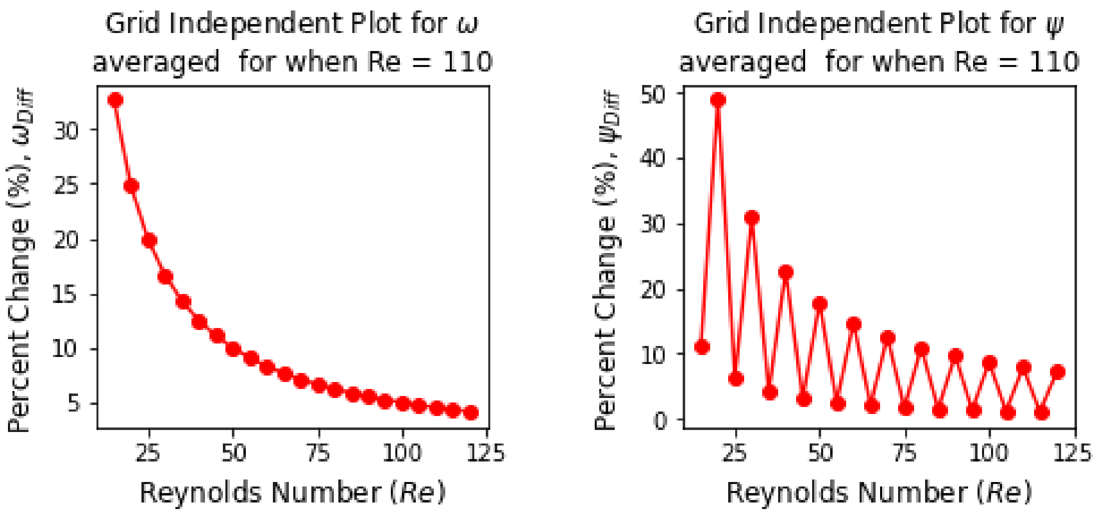
[17]:

```
############# PLOT RESULTS CASE 3 GRID INDEPENDENCE ###################
# Plot of grid independence from results where Re is independent at
# Re = 110

# Understand and interpret data to check reynolds number
# Take average of vorticity and streamline for whole contour plot in
# this case

omega_diff1 = np.zeros(len(k)-1) # vorticity
psi_diff1 = np.zeros(len(k)-1) # streamfunction


for r in range(0,len(k)-1):

    omega1, omega_avg1, psi1, psi_avg1, I, e_Vort, e_Stream, iteration, dx, l  =␣
 ↪test_5[r]
    omega2, omega_avg2, psi2, psi_avg2, I, e_Vort, e_Stream, iteration, dx, l  =␣
 ↪test_5[r+1]

    omega_diff1[r] = ((abs(omega_avg1 - omega_avg2)) / abs(omega_avg1 )) * 100
    psi_diff1[r] = ((abs(psi_avg1 - psi_avg2)) / psi_avg1 ) * 100


# Plots
##############################################################################  ␣
 ↪
# Convergence Plot (Change in Vorticity vs. Grid Number)
fsy=6
fig = plt.figure(figsize = (1.25*fsy, fsy)) # Set the figure size to square
left = 0.1 # the left side of the subplots of the figure
```

```python
right = 0.9 # the right side of the subplots of the figure
bottom = 0.15 # the bottom of the subplots of the figure
top = 0.5 # the top of the subplots of the figure
wspace = 0.5 # the amount of width reserved for space between subplots,
# expressed as a fraction of the average axis width
hspace = .25 # the amount of height reserved for space between subplots,
# expressed as a fraction of the average axis height
plt.subplots_adjust(left, bottom, right, top, wspace, hspace)

plt.subplot(1,2,1)
plt.plot(k[0:-1], omega_diff1, color = 'r', marker = 'o')
plt.xlabel('Reynolds Number $(Re)$', fontsize =12)
plt.ylabel('Percent Change (%), $\omega_{Diff}$ ', fontsize =12)
plt.title('Grid Independent Plot for $\omega$ \naveraged  for when Re = 110')

# Convergence Plot (Change in Streamfunction vs. Number)
plt.subplot(1,2,2)
plt.plot(k[0:-1], psi_diff1, color = 'r', marker = 'o')
plt.xlabel('Reynolds Number $(Re)$', fontsize =12)
plt.ylabel('Percent Change (%), $\psi_{Diff}$ ', fontsize =12)
plt.title('Grid Independent Plot for $\psi$ \naveraged  for when Re = 110')
plt.show()

############################################################################
# now check convergence based on change in omega For vorticity
for l in range(0, len(omega_diff1)):

    if abs(omega_diff1[l]) < 5.0:
        break


Num_grid_points = k[l]
print('For a grid independent solution with less than 5.0% percent change in␣
 ↪streamfunction requires, '
      'Number of grid points =', Num_grid_points,'\n for a grid independent␣
 ↪solution. ' )

# now check convergence based on change in omega For streamfunction
for l in range(0, len(psi_diff1)):

    if abs(psi_diff1[l]) < 5.0:
        break


Num_grid_points = k[l]
print('For a grid independent solution with less than 5.0% percent change in␣
 ↪streamfunction requires, '
```

```
    'Number of grid points =', Num_grid_points,'\n for a grid independent␣
↪solution. ' )
```

4.2.2 Results




```
For a grid independent solution with less than 5.0% percent change in
streamfunction requires, Number of grid points = 100 for a grid independent␣
↪solution.

For a grid independent solution with less than 5.0% percent change in
streamfunction requires, Number of grid points = 35 for a grid independent␣
↪solution.
```

Looking at the results and choosing the higher number of grid points as the grid independence solution. The observation can be made that grid independence occurs for a 100 grid points. Next we will look at the solution for 100 grids.

[18]:
```
############ CASE 3 Grid INDEPENDENCE FOR HIGHER REYNOLDS NUMBERS ############
import numpy as np

# run function #(I, Re, tol, sigma_Vort, sigma_ Stream) #0.2,10,1.31
test2 = solver(101, 110, 1e-5,  0.034, 1.489)
omega, omega_avg, psi, psi_avg, I, e_Vort, e_Stream, iteration, dx, l = test2

############################# PRINT ##########################################

print("\n\nFor Re = 110, and I =", I, " grid divisions. The solution has␣
↪converged in", iteration, " iterations")
```

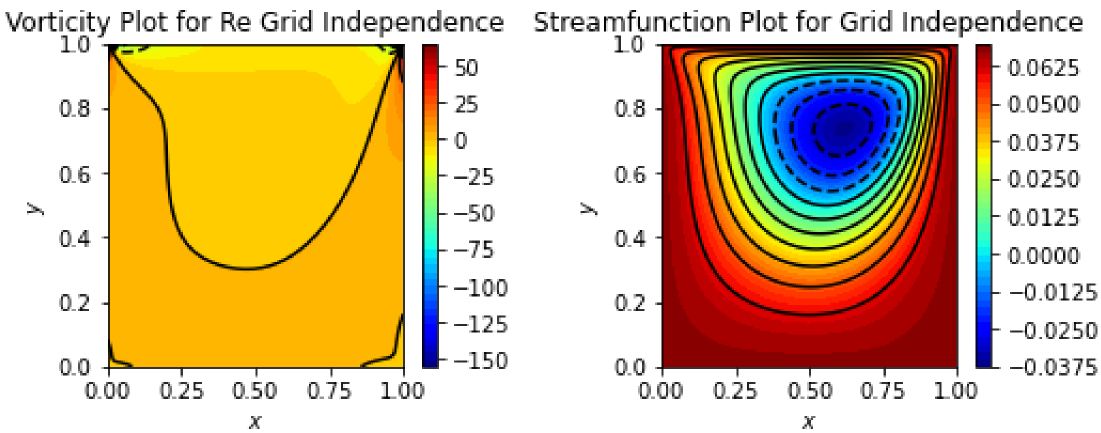
```
print("The convergence criterion of e_omega = ", e_Vort, " for vorticity, and␣
↪e_psi = ", e_Stream, " for streamfunction was achieved ")
```

```
For Re = 0, and I = 101  grid divisions. The solution has converged in 1287
iterations
The convergence criterion of e_omega =  1.6555679831015428e-07  for vorticity,
and e_psi =  9.978726872391717e-06  for streamfunction was achieved
```

[19]:
```
########################## PLOT RESULTS CASE 3 #########################
import matplotlib.pyplot as plt
import scipy.constants

N = I+1
l = 1.
delta = l / I
x = np.linspace(0, l, N)
y = np.linspace(0, l, N)
X, Y = np.meshgrid(x, y)

fsy=6
fig = plt.figure(figsize = (1.25*fsy, fsy)) # Set the figure size to square
left = 0.1 # the left side of the subplots of the figure
right = 0.9 # the right side of the subplots of the figure
bottom = 0.15 # the bottom of the subplots of the figure
top = 0.5 # the top of the subplots of the figure
wspace = 0.5 # the amount of width reserved for space between subplots,
# expressed as a fraction of the average axis width
hspace = .25 # the amount of height reserved for space between subplots,
# expressed as a fraction of the average axis height
plt.subplots_adjust(left, bottom, right, top, wspace, hspace)


# plot vorticity
plt.subplot(1,2,1)
plt.contourf(Y, X, omega, 50, cmap = 'jet')
plt.colorbar()
plt.contour(Y, X, omega, 10, colors = 'k')
plt.title('Vorticity Plot for Re Grid Independence')
plt.xlabel("$x$")
plt.ylabel("$y$")


# plot streamfunction
plt.subplot(1,2,2)
plt.contourf(Y, X, psi, 50, cmap = 'jet')
plt.colorbar()
```

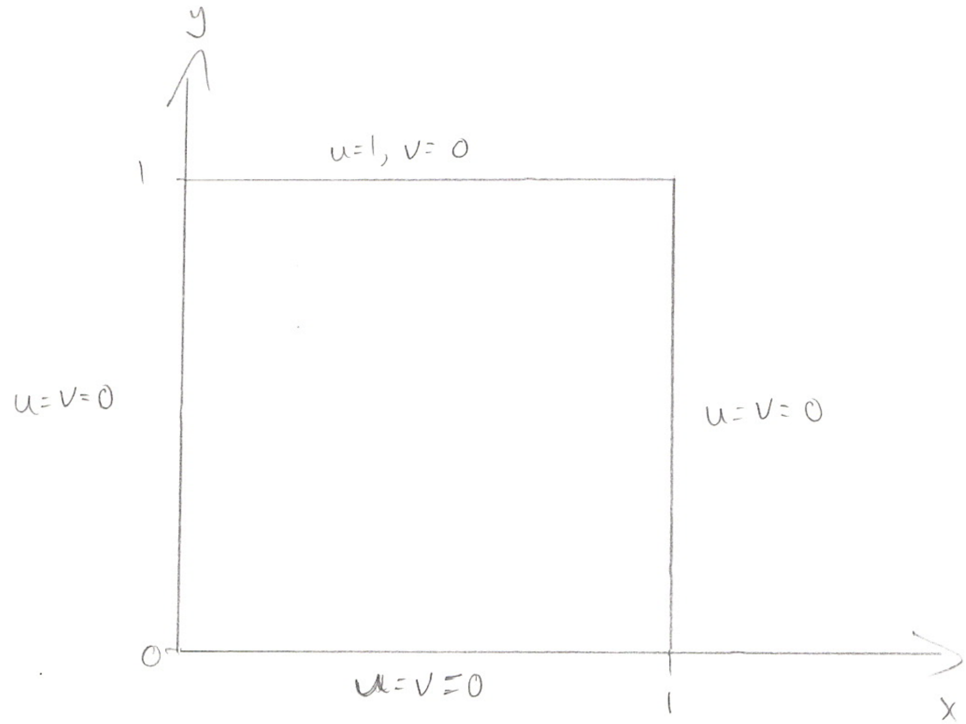
```
plt.contour(Y, X, psi, 10, colors = 'k')
plt.title('Streamfunction Plot for Grid Independence')
plt.xlabel("$x$")
plt.ylabel("$y$")

plt.show()
```

4.3 Plots of Grid Independent Solution



5 Derivations

Below are the derivations and written work done in order to have a better understanding of this problem. The functions were developed in problem set #3 and #4 however written work was still done.



All 4 boundaries are dirichlet

↑ don't need to be updated

vorticity transport eqn:

$$u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right)$$

poisson eqn

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega$$ where BC is $\psi = 0$ for all boundaries

use central difference approximations to evaluate $u = \frac{\partial \psi}{\partial y}$, $v = -\frac{\partial \psi}{\partial x}$ to obtain updated velocities $u(x,y)$ and $v(x,y)$

$$\frac{\ }{\partial x} \quad Re \left[\frac{\qquad}{\Delta y^2} \right] - \frac{\ }{\Delta x} \left\{ w_{i+1,j} - w_{i,j},\ w^*_{i,j} < 0 \right.$$

multiply by Δy^2 to keep coefficients $O(1)$

$$\Delta x^2 T_x = \begin{cases} \frac{1}{Re}\left[w_{i+1,j} - (2-\sigma) w_{i,j} + w_{i-1,j} \right] - w^*_{i,j} \Delta x \left[w_{i,j} - w_{i-1,j} \right], & w^*_{i,j} > 0 \\ \frac{1}{Re}\left[w_{i+1,j} - (2-\sigma) w_{i,j} + w_{i+1,j} \right] - w^*_{i,j} \Delta x \left[w_{i+1,j} - w_{i,j} \right], & w^*_{i,j} < 0 \end{cases}$$

group like terms

$$\Delta x^2 T_x = \begin{cases} \left[\frac{1}{Re}\right] w_{i+1,j} + \left[\frac{-2-\sigma}{Re} - w^*_{i,j}\Delta x\right] w_{i,j} + \left[\frac{1}{Re} + w^*_{i,j}\Delta x\right] w_{i-1,j}, & w^*_{i,j} > 0 \\ \left[\frac{1}{Re} - w^*_{i,j}\Delta x\right] w_{i+1,j} + \left[\frac{-2-\sigma}{Re} + w^*_{i,j}\Delta x\right] w_{i,j} + \left[\frac{1}{Re}\right] w_{i-1,j}, & w^*_{i,j} < 0 \end{cases}$$

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i$$

for $u^*_{i,j} > 0$

$$a = \frac{1}{Re} + w^*_{i,j}\Delta x$$

$$b = \frac{-2}{Re} - \sigma - \Delta x\, w^*_{i,j}$$

$$c = \frac{1}{Re}$$

for $u^*_{i,j} < 0$

$$a = \frac{1}{Re}$$

$$b = \frac{-2}{Re} - \sigma + \Delta x\, w^*_{i,j}$$

$$c = \frac{1}{Re} - \Delta x\, w^*_{i,j}$$

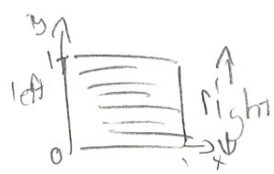
for $u^*_{i,j} = 0$

$$a = \frac{1}{Re}$$

$$b = \frac{-2}{Re} - \sigma$$

$$c = \frac{1}{Re}$$

$BC = [\,p \quad q \quad r\,]$



constant y-lines

Left $BC = [\,1 \quad 0 \quad w_{0,j}\,]$

Right $B = [\,1 \quad 0 \quad w_{N,j}\,]$

from PS #3 α is used

so for PS #4 $\alpha = \frac{1}{Re} \Rightarrow Re = \frac{1}{\alpha}$

$$u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right)$$

$$T_x = \frac{1}{Re}\frac{\partial^2 \omega}{\partial x^2} - u\frac{\partial \omega}{\partial x} = v\frac{\partial \omega}{\partial y} - \frac{1}{Re}\frac{\partial^2 \omega}{\partial y^2}$$

solve for RHS

$$v\frac{\partial \omega}{\partial y} - \frac{1}{Re}\frac{\partial^2 \omega}{\partial y^2} = \frac{\omega^*_{i,j}}{\Delta y}[?] - \frac{1}{Re}\left[\frac{\omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j}}{\Delta y^2}\right] - \frac{\sigma}{\Delta y^2}\omega_{i,j}$$

remember T_x is multiplied by Δx^2

$$\Delta x^2 T_x = \frac{\omega^*_{i,j}}{\Delta y}[?]\Delta x^2 - \frac{\Delta x^2}{Re}\left[\frac{\omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j}}{\Delta y^2}\right] - \frac{\sigma}{\Delta y^2}\omega_{i,j}\Delta x^2$$

$$\frac{\Delta x}{\Delta y} = \bar{\Delta} = 1$$

$$\Delta x^2 T_x = \omega^*_{i,j}\Delta x[?] - \frac{1}{Re}[\omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j}] - \sigma\omega_{i,j}$$

---

for $\omega^*_{i,j} > 0$ $\quad \Delta x^2 T_x = d = \omega^*_{i,j}\Delta x[\omega_{i,j} - \omega_{i-1,j}] - \ldots$

$$d = \left[-\frac{1}{Re}\right]\omega_{i+1,j} + \left[\omega^*_{i,j}\Delta x + \frac{2}{Re} - \sigma\right]\omega_{i,j} + \left[\omega^*_{i,j}\Delta x - \frac{1}{Re}\right]\omega_{i,j}$$

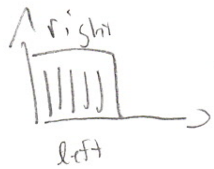
for $\omega^*_{i,j} < 0$

$$d = \left[\omega^*_{i,j}\Delta x - \frac{1}{Re}\right]\omega_{i+1,j} + \left[-\omega^*_{i,j}\Delta x + \frac{2}{Re} - \sigma\right]\omega_{i,j} + \left[-\frac{1}{Re}\right]\omega_{i-1,j}$$

for $\omega^*_{i,j} = 0$

$$d = \left[-\frac{1}{Re}\right]\omega_{i+1,j} + \left[\frac{2}{Re} - \sigma\right]\omega_{i,j} + \left[-\frac{1}{Re}\right]\omega_{i-1,j}$$

Same eqn's as before for a, b, c, d when solving

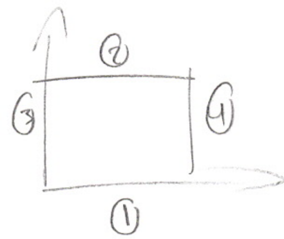


constant x-lines

Left BC = $[1, 0, \omega_{i,0}]$

right BC = $[1, 0, \omega_{i,N-1}]$

Look at BCs for streamfunction



2nd order central difference

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + O(\Delta x^2)$$

① lower BC

$$-\omega = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}$$

↳ $$-\omega_{i,0} = \frac{\psi_{i,1} - 2\psi_{i,0} + \psi_{i,-1}}{\Delta x^2}$$

out of bound $\psi_{i,-1} = \psi_{i,1}$

$$-\omega_{i,0} = \frac{(2)[\psi_{i,1} - \psi_{i,0}]}{\Delta y^2}$$

$$\omega_{i,0} = \frac{2}{\Delta x^2}[\psi_{i,0} - \psi_{i,1}]$$

② upper BC

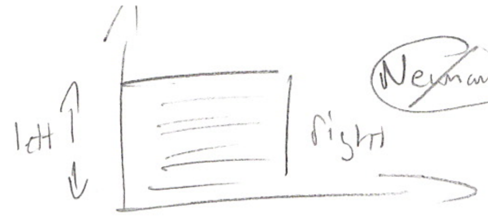
$$\omega_{i,N} = \frac{2}{\Delta x^2}[\psi_{i,N} - \psi_{i,N-1} - \Delta x]$$

③ left BC

$$\omega_{0,j} = \frac{2}{\Delta x^2}[\psi_{0,j} - \psi_{1,j}]$$

④ right BC

$$\omega_{N,j} = \frac{2}{\Delta x^2}[\psi_{j,N} - \psi_{j,N-1}]$$



Neuman

dirchlet boundaries now

$\text{Left} = [1 \quad 0 \quad u_{0,j}]$

$\text{right} = [1 \quad 0 \quad u_{N,j}]$

Constant y-lines

(slide 19?) $\underset{C}{u_{i+1,j}} - \underset{B}{(2+\sigma)u_{i,j}} + \underset{A}{u_{i-1,j}} = \Delta x^2 f_{i,j} - \Delta[u_{i,j+1} - (2 - \frac{\sigma}{\Delta})u_{i,j} + u_{i,j-1}], \; i = 2, \dots I$

$A = 1$

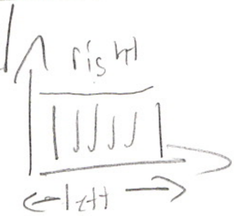
$B = -2 - \sigma$

$C = 1$

$$d = \Delta x^2 f_{i,j} - [u_{i,j+1} - (2-\sigma)u_{i,j} + u_{i,j-1}], \; i = 2 \dots I$$

for $j \neq$ boundaries since dirchlet

Costant x-lines



$\text{left} = [1, 0, u_{i,0}]$

$\text{right} = [1, 0, u_{i,N}]$

$A = 1$

$B = -2 - \sigma$

$C = 1$

$$d = \Delta x^2 f_{i,j} - [u_{i+1,j} - (2-\sigma)u_{i,j} + u_{i-1,j}], \; j = 2 \dots J$$