# 1 PSet-6.py

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1.axes_divider import make_axes_locatable
import matplotlib.patches as patches

import re
import cv2
import os

import time
start_time = time.time()



CURRENT = slice(1, -1),   slice(1, -1)
LEFT    = slice(0, -2),   slice(1, -1)
RIGHT   = slice(2, None), slice(1, -1)
DOWN    = slice(1, -1),   slice(0, -2)
UP      = slice(1, -1),   slice(2, None)


fileName = "Final-Project"

final_frame_only = True
generate_video = False

height = 200
width = 500

num_time_steps = 100

cylinder_diameter = 50
cylinder_radius = cylinder_diameter / 2
```

```
cylinder_center = [(height / 2), 100]

error_limit = 0.01                    # 1% maximum change for convergence

U_inf = 2                             # m/s uniform inflow
F = 1.9                               # over-relaxation factor
free_lid = U_inf * (height / 2)       # free-lid streamfunction constant

Re_D = 200                            # Given Reynolds number

T_surface = 400                       # K
T_boundary = 300                      # K
T_init = min(T_surface, T_boundary) # Bulk fluid initial temp

# Constants picked for air around room temp
alpha = 22.07 * 10**(-6)    # m^2/s     Thermal Diffusivity at 300K
nu = 1.48 * 10**(-5)        # m^2/s     Kinematic Viscosity at 300K


h_1 = (10 - 1) * nu / U_inf
h_2 = (10 - 1) * alpha / U_inf
h = min(h_1, h_2)       # grid spacing
dt = (h / U_inf) / 2



###############################################################
#  Setup Grid Points and Solid Body
###############################################################
omega = np.zeros((width, height))               # vorticity
psi = np.zeros((width, height))                 # streamfunction
temps = np.zeros((width, height)) + T_init      # temperature


solid_rows = []
```

```
solid_cols = []

def solid_body_setup(width, height):
    for i in range(width):
        for j in range(height):
            dist = np.sqrt((i - cylinder_center[0])**2 + (j - cylinder_center[1])**2)
            if (dist <= cylinder_radius):
                solid_rows.append(i)
                solid_cols.append(j)
    solid_points = list(zip(solid_rows, solid_cols))
solid_body_setup(width, height)

solid_points = list(zip(solid_rows, solid_cols))

def test_solid_setup():
    solid_body_test = np.zeros((width, height))
    solid_body_test[solid_rows, solid_cols] = 1


    figNum = 1
    fig = plt.figure(figNum)
    plt.axes().set_aspect('equal')
    data_graphable = np.flipud(np.rot90(solid_body_test))

    plt.pcolor(data_graphable)

    plt.show()
# test_solid_setup()


wall_rows = []
wall_cols = []
wall_adj_rows = []
wall_adj_cols = []

```

```python
def wall_setup():
    count = 0
    for i in range(width):
        for j in range(height):
            if (i, j) in solid_points:
                count += 1
                if (i - 1, j) not in solid_points:
                    wall_rows.append(i)
                    wall_cols.append(j)
                    wall_adj_rows.append(i - 1)
                    wall_adj_cols.append(j)
                elif (i + 1, j) not in solid_points:
                    wall_rows.append(i)
                    wall_cols.append(j)
                    wall_adj_rows.append(i + 1)
                    wall_adj_cols.append(j)
                elif (i, j - 1) not in solid_points:
                    wall_rows.append(i)
                    wall_cols.append(j)
                    wall_adj_rows.append(i)
                    wall_adj_cols.append(j - 1)
                elif (i, j + 1) not in solid_points:
                    wall_rows.append(i)
                    wall_cols.append(j)
                    wall_adj_rows.append(i)
                    wall_adj_cols.append(j + 1)
wall_setup()

def test_wall_setup():
    wall_test = np.zeros((width, height))
    wall_test[wall_rows, wall_cols] = 1
    # wall_test[solid_rows, solid_cols] = 1
    wall_test[wall_adj_rows, wall_adj_cols] = 2


```

```
    figNum = 4
    fig = plt.figure(figNum)
    plt.axes().set_aspect('equal')
    data_graphable = np.flipud(np.rot90(wall_test))

    plt.pcolor(data_graphable)

    plt.show()
# test_wall_setup()

bulk_rows = []
bulk_cols = []

def bulk_setup():
    for i in range(1, width - 1):
        for j in range(1, height - 1):
            if (i, j) not in solid_points:
                bulk_rows.append(i)
                bulk_cols.append(j)
bulk_setup()

bulk_points = list(zip(bulk_rows, bulk_cols))

def test_bulk_setup():
    bulk_test = np.zeros((width, height))
    bulk_test[bulk_rows, bulk_cols] = 1


    figNum = 5
    fig = plt.figure(figNum)
    plt.axes().set_aspect('equal')
    data_graphable = np.flipud(np.rot90(bulk_test))

    plt.pcolor(data_graphable)

```

```python
    plt.show()
# test_bulk_setup()


def gauss_seidel_iteration(data, initial = False):
    """
    Perform Gauss-Seidel Iteration

    @param data: 2D array (width, height) of values to be relaxed by Gauss-Seidel Iteration
    @param initial: Determines which Poisson/Laplacian equation will be used.

    @return: data array post-relaxation iteration (same dimensions as @param data).
    """
    error_flag = True
    while error_flag:
        data_old = data.copy()

        # data[i, j] = data[i, j] + (F / 4) * (data[i + 1, j] + data[i - 1, j] + data[i, j + 1] + data[i, j - 1] - 4 * data[i, j])
        # data[1:-1, 1:-1] = data[1:-1, 1:-1] + (1 / 4) * (data[0:-2, 1:-1] + data[2:, 1:-1] + data[1:-1,0:-2] + data[1:-1, 2:] - 4 * data[1:-1, 1:-1])

        data[CURRENT] = data[CURRENT] + (1 / 4) * (data[LEFT] + data[RIGHT] + data[DOWN] + data[UP] - 4 * data[CURRENT])
        if not initial:
            data[CURRENT] = data[CURRENT] + h * h * omega[CURRENT]   # Multiply by F

        data[0, :] = data[3, :]
        data[width - 1, :] = data[width - 2, :]


        data[solid_rows, solid_cols] = 0


```

```
        data_abs_diff = np.absolute(data - data_old)
        error_array = np.divide(data_abs_diff, data_old, out = np.zeros_like(data), where = ((data_abs_diff != 0) & (data_old != 0)))
        error_array[error_array == np.inf] = 0
        error_term = np.amax(error_array)

        if (error_term <= error_limit):
            error_flag = False

    return data



##############################################################
#  Initial Conditions
##############################################################
psi[solid_rows, solid_cols] = 0
temps[solid_rows, solid_cols] = T_surface

psi[:, cylinder_center[1]] = 0
psi[:, 0] = -free_lid
psi[:, (height - 1)] = free_lid

for (i, j) in bulk_points:
    psi[i, j] = U_inf * j - free_lid

psi = gauss_seidel_iteration(psi, initial = True)

print("--- Initial Psi Setup ---")
print("--- %.7f seconds ---\n" % (time.time() - start_time))


def test_initial_setup():
    figNum = 2
    fig = plt.figure(figNum)
```

```
    plt.axes().set_aspect('equal')
    data_graphable = np.flipud(np.rot90(psi))


    num_streamlines = 31
    max_streamline = np.max(data_graphable)
    min_streamline = np.min(data_graphable)
    contours_before = np.linspace(min_streamline, max_streamline, num=(num_streamlines + 3))
    contours = contours_before[(contours_before != 0) & (contours_before != min_streamline) & (contours_before != max_streamline)]

    plt.contour(data_graphable, levels = contours, colors = 'black', linestyles = 'solid')


    plt.xlim(0, width)
    plt.ylim(0, height)
    plt.xticks(np.arange(0, width + 1, 50))
    plt.yticks(np.arange(0, height + 1, 20))
    plt.tick_params(top=True, right=True)

    plt.style.use('grayscale')
    heatmap = plt.pcolor(data_graphable)
    plt.clim(np.amin(data_graphable), np.amax(data_graphable))

    plt.show()
# test_initial_setup()

print("Time Step: 1 of " + str(num_time_steps))



##############################################################
#  Initial Conditions established, now "turn on" vorticity
```

```python
################################################################
u = np.zeros((width, height))
v = np.zeros((width, height))

omega_history = [omega.copy()]
psi_history = [psi.copy()]
temps_history = [temps.copy()]

wall_rows_left = [x - 1 for x in wall_rows]
wall_rows_right = [x + 1 for x in wall_rows]
wall_cols_down = [y - 1 for y in wall_cols]
wall_cols_up = [y + 1 for y in wall_cols]

# bulk_rows_left = [x - 1 for x in bulk_rows]
# bulk_rows_right = [x + 1 for x in bulk_rows]
# bulk_cols_down = [y - 1 for y in bulk_cols]
# bulk_cols_up = [y + 1 for y in bulk_cols]

# u_delta_T = np.zeros((width, height))
# v_delta_T = np.zeros((width, height))
# temps_laplacian = np.zeros((width, height))

delta_u_omega = np.zeros((width, height))
delta_v_omega = np.zeros((width, height))

vorticity_laplacian = np.zeros((width, height))

u_delta_T = np.zeros((width, height))
v_delta_T = np.zeros((width, height))
temps_laplacian = np.zeros((width, height))

bulk_rows_left = [x - 1 for x in bulk_rows]
bulk_rows_right = [x + 1 for x in bulk_rows]
bulk_cols_down = [y - 1 for y in bulk_cols]
bulk_cols_up = [y + 1 for y in bulk_cols]
```

```


for n in range(1, num_time_steps):
    omega_prev = omega.copy()
    temps_prev = temps.copy()

    # omega[wall_rows][wall_cols] = -2 * (psi[wall_adj_rows][wall_adj_cols] - psi[
    wall_rows][wall_cols]) / (h * h)

    omega[wall_rows, wall_cols] = -2 / (h * h) * (psi[wall_rows_right, wall_cols] + psi[
    wall_rows_left, wall_cols] + psi[wall_rows, wall_cols_up] + psi[wall_rows,
    wall_cols_down])


    u.fill(0)
    v.fill(0)

    ### Method 2 (Slowly):

    # for (i, j) in bulk_points:
    #     u[i, j] = (psi[i, j + 1] - psi[i, j - 1]) / (2 * h)
    #     v[i, j] = (psi[i - 1, j] - psi[i + 1, j]) / (2 * h)


    # for (i, j) in bulk_points:
    #     delta_u_omega = 0

    #     if (u[i, j] < 0):
    #         delta_u_omega = u[i + 1, j] * omega_prev[i + 1, j] - u[i, j] * omega_prev[
    i, j]
    #     elif (u[i, j] > 0):
    #         delta_u_omega = u[i, j] * omega_prev[i, j] - u[i - 1, j]

    #     delta_v_omega = 0
    #     if (v[i, j] < 0):
```

```
    #         delta_v_omega = v[i, j + 1] * omega_prev[i, j + 1] - v[i, j] * omega_prev[i, j]
    #     elif (v[i, j] > 0):
    #         delta_v_omega = v[i, j] * omega_prev[i, j] - v[i, j - 1] * omega_prev[i, j - 1]

    #     vorticity_laplacian = (omega_prev[i + 1, j] + omega_prev[i - 1, j] + omega_prev[i, j + 1] + omega_prev[i, j - 1] - 4 * omega_prev[i, j]) / (h * h)

    #     omega[i, j] = omega_prev[i, j] + dt * (-delta_u_omega / h - delta_v_omega / h + nu * vorticity_laplacian)



    #     u_delta_T = 0
    #     if (u[i, j] < 0):
    #         u_delta_T = u[i, j] * (temps_prev[i + 1, j] - temps_prev[i, j])
    #     elif (u[i, j] > 0):
    #         u_delta_T = u[i, j] * (temps_prev[i, j] - temps_prev[i - 1, j])

    #     v_delta_T = 0
    #     if (v[i, j] < 0):
    #         v_delta_T = v[i, j] * (temps_prev[i, j + 1] - temps_prev[i, j])
    #     elif (v[i, j] > 0):
    #         v_delta_T = v[i, j] * (temps_prev[i, j] - temps_prev[i, j - 1])

    #     temps_laplacian = (temps_prev[i - 1, j] + temps_prev[i + 1, j] + temps_prev[i, j - 1] + temps_prev[i, j + 1] - 4 * temps_prev[i, j]) / (h * h)

    #     temps[i, j] = temps_prev[i, j] + dt * ((-u_delta_T - v_delta_T) / h + alpha * temps_laplacian)

    #     temps[i, j] = (temps_prev[i - 1, j] + temps_prev[i + 1, j] + temps_prev[i, j - 1] + temps_prev[i, j + 1]) / 4

```

```python



    # psi = gauss_seidel_iteration(psi)


    ### End Method 2


    print(np.amax(omega))

    ### Method 1 (Fast):
    # u_delta_T.fill(0)
    # v_delta_T.fill(0)

    # u_neg_ind = np.nonzero(u < 0)
    # u_pos_ind = np.nonzero(u > 0)
    # v_neg_ind = np.nonzero(v < 0)
    # v_pos_ind = np.nonzero(v > 0)

    # u_neg_ind_right = u_neg_ind[:]
    # u_neg_ind_right[:][1][:] += 1         # Should this be 0?

    # u_pos_ind_left = u_pos_ind[:]
    # u_pos_ind_left[:][1][:] += -1         # Should this be 0?


    # v_neg_ind_up = v_neg_ind[:]
    # v_neg_ind_up[:][0][:] += 1            # Should this be 1?

    # v_pos_ind_down = v_pos_ind[:]
    # v_pos_ind_down[:][0][:] += -1         # Should this be 1?


    # delta_u_omega[u_neg_ind] = u[u_neg_ind_right] * omega[u_neg_ind_right] - u[
```

```
u_neg_ind] * omega[u_neg_ind]
    # delta_u_omega[u_pos_ind] = u[u_pos_ind] * omega[u_pos_ind] - u[u_pos_ind_left] *
omega[u_pos_ind_left]

    # delta_v_omega[v_neg_ind] = v[v_neg_ind_up] * omega[v_neg_ind_up] - v[v_neg_ind] *
omega[v_neg_ind]
    # delta_v_omega[v_pos_ind] = v[v_pos_ind] * omega[v_pos_ind] - v[v_pos_ind_down] *
omega[v_pos_ind_down]

    # vorticity_laplacian[CURRENT] = (omega[UP] + omega[DOWN] + omega[LEFT] + omega[
RIGHT] - 4 * omega[CURRENT]) / (h * h)

    # omega[CURRENT] += dt * (-delta_u_omega[CURRENT] - delta_v_omega[CURRENT]) / h + nu
 * vorticity_laplacian[CURRENT]





    # psi = gauss_seidel_iteration(psi)





    # u_delta_T[u_neg_ind] = u[u_neg_ind] * (temps_prev[u_neg_ind_right] - temps_prev[
u_neg_ind])
    # u_delta_T[u_pos_ind] = u[u_pos_ind] * (temps_prev[u_pos_ind] - temps_prev[
u_pos_ind_left])

    # v_delta_T[v_neg_ind] = v[v_neg_ind] * (temps_prev[v_neg_ind_up] - temps_prev[
v_neg_ind])
    # v_delta_T[v_pos_ind] = v[v_pos_ind] * (temps_prev[v_pos_ind] - temps_prev[
v_pos_ind_down])

```

```
    # temps_laplacian[CURRENT] = (temps_prev[UP] + temps_prev[DOWN] + temps_prev[LEFT] +
    temps_prev[RIGHT] - 4 * temps_prev[CURRENT]) / (h * h)

    # temps = temps_prev + dt * ((-u_delta_T - v_delta_T) / h + alpha * temps_laplacian)

    # temps[solid_rows, solid_cols] = T_surface
    # temps[:, 0] = T_boundary
    # temps[:, (height - 1)] = T_boundary


    ### End Method 1



    # Outflow Boundary Conditions:
    psi[width - 1, :] = 2 * psi[width - 2, :] - psi[width - 3, :]
    omega[width - 1, :] = omega[width - 2, :]



    # if ((n + 1) % 10 == 0):
    #     print("Time Step: " + str(n) + " of " + str(num_time_steps))

    print("Time Step: " + str(n + 1) + " of " + str(num_time_steps))



    omega_history.append(omega.copy())
    psi_history.append(psi.copy())
    temps_history.append(temps.copy())


print("\n--- Time Steps Done ---")
print("--- %.6f seconds ---\n" % (time.time() - start_time))

```

```python


def print_data_in_console():
    """ Print the data in the console (readable format) """
    print(np.rot90(psi))
# print_data_in_console()


##############################################################
#  Graphs and Plots
##############################################################
def delete_previous_images():
    image_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project\\images"

    old_images = [old_img for old_img in os.listdir(image_folder) if old_img.endswith(".png") ]
    for old_image in old_images:
        os.remove(os.path.join(image_folder, old_image))
delete_previous_images()


fig = plt.figure(figsize=(10, 10.5))

for plot_index in range(num_time_steps):
    if final_frame_only and (plot_index != num_time_steps - 1):
        continue

    figNum = plot_index

    ##############################################################
    #  Streamfunction Plot
    ##############################################################
    sub1 = plt.subplot(3, 1, 1, aspect = 'equal')
    data_graphable = np.flipud(np.rot90(psi_history[plot_index]))
```

```
    plt.title("Streamfunction")

    num_streamlines = 31
    max_streamline = np.max(data_graphable)
    min_streamline = np.min(data_graphable)
    contours_before = np.linspace(min_streamline, max_streamline, num=(num_streamlines + 3))
    contours = contours_before[(contours_before != 0) & (contours_before != min_streamline) & (contours_before != max_streamline)]

    plt.contour(data_graphable, levels = contours, colors = 'black', linestyles = 'solid')


    plt.style.use('grayscale')
    plt.xticks(np.arange(0, width + 1, 50))
    plt.yticks(np.arange(0, height + 1, 20))
    plt.tick_params(top=True, right=True)

    plt.pcolor(data_graphable)

    # Color bar:
    divider1 = make_axes_locatable(sub1)
    cax1 = divider1.append_axes('right', size = '3%', pad = 0.3)
    im = sub1.imshow(data_graphable, origin = 'lower', aspect = 'equal', interpolation = 'none')
    fig.colorbar(im, cax = cax1, orientation = 'vertical')




    ################################################################
    #  Vorticity Plot
    ################################################################
```

```
    sub2 = plt.subplot(3, 1, 2, aspect = 'equal')
    data_graphable = np.flipud(np.rot90(omega_history[plot_index]))

    plt.title("Vorticity")
    plt.axis("off")

    plt.pcolor(data_graphable)

    # Color bar:
    divider2 = make_axes_locatable(sub2)
    cax2 = divider2.append_axes('right', size = '3%', pad = 0.3)
    im = sub2.imshow(data_graphable, origin = 'lower', aspect = 'equal', interpolation = 'none')
    fig.colorbar(im, cax = cax2, orientation = 'vertical')




    ###############################################################
    #  Temperature Plot
    ###############################################################
    sub3 = fig.add_subplot(3, 1, 3, aspect = 'equal')
    data_graphable = np.flipud(np.rot90(temps_history[plot_index]))
    plt.title("Temperature")

    plt.style.use('classic')
    plt.axis("off")

    # Color bar:
    divider3 = make_axes_locatable(sub3)
    cax3 = divider3.append_axes('right', size = '3%', pad = 0.3)
    im = sub3.imshow(data_graphable, origin = 'lower', aspect = 'equal', interpolation = 'none')
    cbar = fig.colorbar(im, cax=cax3, orientation = 'vertical')
    cbar.set_label("Temperature (\N{DEGREE SIGN}C)")
```

```



    ################################################################
    #  Time Stamp
    ################################################################
    realtime = plot_index * dt
    realtime_str = "Time: " +  "{:.10f}".format(realtime) + " s"
    fig.suptitle(realtime_str, y = 0.07)



    ################################################################
    #  Save Frame
    ################################################################
    plt.savefig(fileName + "/images/" + fileName + "-Figure-" + str(figNum + 1) + ".png"
   )
    plt.clf()

    print("Frame: " + str(figNum + 1) + " of " + str(num_time_steps))

print("\n--- Figures Done ---")
print("--- %.6f seconds ---" % (time.time() - start_time))


################################################################
#  Generate Video
################################################################
def atoi(text):
    # https://stackoverflow.com/questions/5967500/how-to-correctly-sort-a-string-with-a-
   number-inside
    return int(text) if text.isdigit() else text

def natural_keys(text):
    # https://stackoverflow.com/questions/5967500/how-to-correctly-sort-a-string-with-a-
```

```
    number-inside
    return [ atoi(c) for c in re.split(r'(\d+)', text) ]

def generate_video():
    image_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project\\images"
    output_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project\\"
    video_name = output_folder + "final-project.mp4"
    fps = 5

    images = [img for img in os.listdir(image_folder) if img.endswith(".png")]
    images.sort(key=natural_keys)
    frame = cv2.imread(os.path.join(image_folder, images[0]))
    height, width, layers = frame.shape

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    video = cv2.VideoWriter(video_name, fourcc, fps, (width, height))

    for image in images:
        video.write(cv2.imread(os.path.join(image_folder, image)))

    cv2.destroyAllWindows()
    video.release()
if generate_video:
    generate_video()

print("\n--- Video Done ---")
print("--- %.6f seconds ---" % (time.time() - start_time))
```