

1 PSet-6.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.axes_grid1.axes_divider import make_axes_locatable
4 import matplotlib.patches as patches
5
6 import re
7 import cv2
8 import os
9
10 import time
11 start_time = time.time()
12
13
14
15 CURRENT = slice(1, -1), slice(1, -1)
16 LEFT    = slice(0, -2), slice(1, -1)
17 RIGHT   = slice(2, None), slice(1, -1)
18 DOWN    = slice(1, -1), slice(0, -2)
19 UP      = slice(1, -1), slice(2, None)
20
21
22 fileName = "Final-Project"
23
24 final_frame_only = True
25 generate_video = False
26
27 height = 200
28 width = 500
29
30 num_time_steps = 100
31
32 cylinder_diameter = 50
33 cylinder_radius = cylinder_diameter / 2
```

```

34 cylinder_center = [(height / 2), 100]
35
36 error_limit = 0.01                # 1% maximum change for convergence
37
38 U_inf = 2                        # m/s uniform inflow
39 F = 1.9                          # over-relaxation factor
40 free_lid = U_inf * (height / 2)  # free-lid streamfunction constant
41
42 Re_D = 200                       # Given Reynolds number
43
44 T_surface = 400                  # K
45 T_boundary = 300                 # K
46 T_init = min(T_surface, T_boundary) # Bulk fluid initial temp
47
48 # Constants picked for air around room temp
49 alpha = 22.07 * 10**(-6)        # m^2/s      Thermal Diffusivity at 300K
50 nu = 1.48 * 10**(-5)            # m^2/s      Kinematic Viscosity at 300K
51
52
53 h_1 = (10 - 1) * nu / U_inf
54 h_2 = (10 - 1) * alpha / U_inf
55 h = min(h_1, h_2)               # grid spacing
56 dt = (h / U_inf) / 2
57
58
59
60 #####
61 #   Setup Grid Points and Solid Body
62 #####
63 omega = np.zeros((width, height)) # vorticity
64 psi = np.zeros((width, height))   # streamfunction
65 temps = np.zeros((width, height)) + T_init # temperature
66
67
68 solid_rows = []

```

```

69 solid_cols = []
70
71 def solid_body_setup(width, height):
72     for i in range(width):
73         for j in range(height):
74             dist = np.sqrt((i - cylinder_center[0])**2 + (j - cylinder_center[1])**2)
75             if (dist <= cylinder_radius):
76                 solid_rows.append(i)
77                 solid_cols.append(j)
78     solid_points = list(zip(solid_rows, solid_cols))
79 solid_body_setup(width, height)
80
81 solid_points = list(zip(solid_rows, solid_cols))
82
83 def test_solid_setup():
84     solid_body_test = np.zeros((width, height))
85     solid_body_test[solid_rows, solid_cols] = 1
86
87
88     figNum = 1
89     fig = plt.figure(figNum)
90     plt.axes().set_aspect('equal')
91     data_graphable = np.flipud(np.rot90(solid_body_test))
92
93     plt.pcolor(data_graphable)
94
95     plt.show()
96 # test_solid_setup()
97
98
99 wall_rows = []
100 wall_cols = []
101 wall_adj_rows = []
102 wall_adj_cols = []
103

```

```

104 def wall_setup():
105     count = 0
106     for i in range(width):
107         for j in range(height):
108             if (i, j) in solid_points:
109                 count += 1
110                 if (i - 1, j) not in solid_points:
111                     wall_rows.append(i)
112                     wall_cols.append(j)
113                     wall_adj_rows.append(i - 1)
114                     wall_adj_cols.append(j)
115                 elif (i + 1, j) not in solid_points:
116                     wall_rows.append(i)
117                     wall_cols.append(j)
118                     wall_adj_rows.append(i + 1)
119                     wall_adj_cols.append(j)
120                 elif (i, j - 1) not in solid_points:
121                     wall_rows.append(i)
122                     wall_cols.append(j)
123                     wall_adj_rows.append(i)
124                     wall_adj_cols.append(j - 1)
125                 elif (i, j + 1) not in solid_points:
126                     wall_rows.append(i)
127                     wall_cols.append(j)
128                     wall_adj_rows.append(i)
129                     wall_adj_cols.append(j + 1)
130 wall_setup()
131
132 def test_wall_setup():
133     wall_test = np.zeros((width, height))
134     wall_test[wall_rows, wall_cols] = 1
135     # wall_test[solid_rows, solid_cols] = 1
136     wall_test[wall_adj_rows, wall_adj_cols] = 2
137
138

```

```

139     figNum = 4
140     fig = plt.figure(figNum)
141     plt.axes().set_aspect('equal')
142     data_graphable = np.flipud(np.rot90(wall_test))
143
144     plt.pcolor(data_graphable)
145
146     plt.show()
147 # test_wall_setup()
148
149 bulk_rows = []
150 bulk_cols = []
151
152 def bulk_setup():
153     for i in range(1, width - 1):
154         for j in range(1, height - 1):
155             if (i, j) not in solid_points:
156                 bulk_rows.append(i)
157                 bulk_cols.append(j)
158 bulk_setup()
159
160 bulk_points = list(zip(bulk_rows, bulk_cols))
161
162 def test_bulk_setup():
163     bulk_test = np.zeros((width, height))
164     bulk_test[bulk_rows, bulk_cols] = 1
165
166
167     figNum = 5
168     fig = plt.figure(figNum)
169     plt.axes().set_aspect('equal')
170     data_graphable = np.flipud(np.rot90(bulk_test))
171
172     plt.pcolor(data_graphable)
173

```

```

174     plt.show()
175 # test_bulk_setup()
176
177
178 def gauss_seidel_iteration(data, initial = False):
179     """
180     Perform Gauss-Seidel Iteration
181
182     @param data: 2D array (width, height) of values to be relaxed by Gauss-Seidel
183     Iteration
184     @param initial: Determines which Poisson/Laplacian equation will be used.
185
186     @return: data array post-relaxation iteration (same dimensions as @param data).
187     """
188     error_flag = True
189     while error_flag:
190         data_old = data.copy()
191
192         # data[i, j] = data[i, j] + (F / 4) * (data[i + 1, j] + data[i - 1, j] + data[i,
193         j + 1] + data[i, j - 1] - 4 * data[i, j])
194         # data[1:-1, 1:-1] = data[1:-1, 1:-1] + (1 / 4) * (data[0:-2, 1:-1] + data[2:,
195         1:-1] + data[1:-1, 0:-2] + data[1:-1, 2:] - 4 * data[1:-1, 1:-1])
196
197         data[CURRENT] = data[CURRENT] + (1 / 4) * (data[LEFT] + data[RIGHT] + data[DOWN]
198         + data[UP] - 4 * data[CURRENT])
199         if not initial:
200             data[CURRENT] = data[CURRENT] + h * h * omega[CURRENT] # Multiply by F
201
202         data[0, :] = data[3, :]
203         data[width - 1, :] = data[width - 2, :]
204
205         data[solid_rows, solid_cols] = 0

```

```

205     data_abs_diff = np.absolute(data - data_old)
206     error_array = np.divide(data_abs_diff, data_old, out = np.zeros_like(data),
where = ((data_abs_diff != 0) & (data_old != 0)))
207     error_array[error_array == np.inf] = 0
208     error_term = np.amax(error_array)
209
210     if (error_term <= error_limit):
211         error_flag = False
212
213     return data
214
215
216
217 #####
218 #   Initial Conditions
219 #####
220 psi[solid_rows, solid_cols] = 0
221 temps[solid_rows, solid_cols] = T_surface
222
223 psi[:, cylinder_center[1]] = 0
224 psi[:, 0] = -free_lid
225 psi[:, (height - 1)] = free_lid
226
227 for (i, j) in bulk_points:
228     psi[i, j] = U_inf * j - free_lid
229
230 psi = gauss_seidel_iteration(psi, initial = True)
231
232 print("--- Initial Psi Setup ---")
233 print("--- %.7f seconds ---\n" % (time.time() - start_time))
234
235
236 def test_initial_setup():
237     figNum = 2
238     fig = plt.figure(figNum)

```

```

239 plt.axes().set_aspect('equal')
240 data_graphable = np.flipud(np.rot90(psi))
241
242
243 num_streamlines = 31
244 max_streamline = np.max(data_graphable)
245 min_streamline = np.min(data_graphable)
246 contours_before = np.linspace(min_streamline, max_streamline, num=(num_streamlines +
247 3))
248 contours = contours_before[(contours_before != 0) & (contours_before !=
min_streamline) & (contours_before != max_streamline)]
249
250 plt.contour(data_graphable, levels = contours, colors = 'black', linestyles = 'solid
251 ')
252
253 plt.xlim(0, width)
254 plt.ylim(0, height)
255 plt.xticks(np.arange(0, width + 1, 50))
256 plt.yticks(np.arange(0, height + 1, 20))
257 plt.tick_params(top=True, right=True)
258
259 plt.style.use('grayscale')
260 heatmap = plt.pcolor(data_graphable)
261 plt.clim(np.amin(data_graphable), np.amax(data_graphable))
262
263 plt.show()
264 # test_initial_setup()
265
266 print("Time Step: 1 of " + str(num_time_steps))
267
268
269 #####
270 # Initial Conditions established, now "turn on" vorticity

```



```

271 #####
272 u = np.zeros((width, height))
273 v = np.zeros((width, height))
274
275 omega_history = [omega.copy()]
276 psi_history = [psi.copy()]
277 temps_history = [temps.copy()]
278
279 wall_rows_left = [x - 1 for x in wall_rows]
280 wall_rows_right = [x + 1 for x in wall_rows]
281 wall_cols_down = [y - 1 for y in wall_cols]
282 wall_cols_up = [y + 1 for y in wall_cols]
283
284 # bulk_rows_left = [x - 1 for x in bulk_rows]
285 # bulk_rows_right = [x + 1 for x in bulk_rows]
286 # bulk_cols_down = [y - 1 for y in bulk_cols]
287 # bulk_cols_up = [y + 1 for y in bulk_cols]
288
289 # u_delta_T = np.zeros((width, height))
290 # v_delta_T = np.zeros((width, height))
291 # temps_laplacian = np.zeros((width, height))
292
293 delta_u_omega = np.zeros((width, height))
294 delta_v_omega = np.zeros((width, height))
295
296 vorticity_laplacian = np.zeros((width, height))
297
298 u_delta_T = np.zeros((width, height))
299 v_delta_T = np.zeros((width, height))
300 temps_laplacian = np.zeros((width, height))
301
302 bulk_rows_left = [x - 1 for x in bulk_rows]
303 bulk_rows_right = [x + 1 for x in bulk_rows]
304 bulk_cols_down = [y - 1 for y in bulk_cols]
305 bulk_cols_up = [y + 1 for y in bulk_cols]

```

```

306
307
308 for n in range(1, num_time_steps):
309     omega_prev = omega.copy()
310     temps_prev = temps.copy()
311
312     # omega[wall_rows][wall_cols] = -2 * (psi[wall_adj_rows][wall_adj_cols] - psi[
313     wall_rows][wall_cols]) / (h * h)
314
315     omega[wall_rows, wall_cols] = -2 / (h * h) * (psi[wall_rows_right, wall_cols] + psi[
316     wall_rows_left, wall_cols] + psi[wall_rows, wall_cols_up] + psi[wall_rows,
317     wall_cols_down])
318
319     u.fill(0)
320     v.fill(0)
321
322     ### Method 2 (Slowly):
323
324     # for (i, j) in bulk_points:
325     #     u[i, j] = (psi[i, j + 1] - psi[i, j - 1]) / (2 * h)
326     #     v[i, j] = (psi[i - 1, j] - psi[i + 1, j]) / (2 * h)
327
328     # for (i, j) in bulk_points:
329     #     delta_u_omega = 0
330
331     #     if (u[i, j] < 0):
332     #         delta_u_omega = u[i + 1, j] * omega_prev[i + 1, j] - u[i, j] * omega_prev[
333     i, j]
334     #     elif (u[i, j] > 0):
335     #         delta_u_omega = u[i, j] * omega_prev[i, j] - u[i - 1, j]
336
337     #     delta_v_omega = 0
338     #     if (v[i, j] < 0):

```

```

337     #         delta_v_omega = v[i, j + 1] * omega_prev[i, j + 1] - v[i, j] * omega_prev[
i, j]
338     #         elif (v[i, j] > 0):
339     #             delta_v_omega = v[i, j] * omega_prev[i, j] - v[i, j - 1] * omega_prev[i, j
- 1]
340
341     #         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)
342
343     #         omega[i, j] = omega_prev[i, j] + dt * (-delta_u_omega / h - delta_v_omega / h
+ nu * vorticity_laplacian)
344
345
346
347     #         u_delta_T = 0
348     #         if (u[i, j] < 0):
349     #             u_delta_T = u[i, j] * (temps_prev[i + 1, j] - temps_prev[i, j])
350     #         elif (u[i, j] > 0):
351     #             u_delta_T = u[i, j] * (temps_prev[i, j] - temps_prev[i - 1, j])
352
353     #         v_delta_T = 0
354     #         if (v[i, j] < 0):
355     #             v_delta_T = v[i, j] * (temps_prev[i, j + 1] - temps_prev[i, j])
356     #         elif (v[i, j] > 0):
357     #             v_delta_T = v[i, j] * (temps_prev[i, j] - temps_prev[i, j - 1])
358
359     #         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)
360
361     #         temps[i, j] = temps_prev[i, j] + dt * ((-u_delta_T - v_delta_T) / h + alpha *
temps_laplacian)
362
363     #         temps[i, j] = (temps_prev[i - 1, j] + temps_prev[i + 1, j] + temps_prev[i, j -
1] + temps_prev[i, j + 1]) / 4
364

```

```

365
366
367
368     # psi = gauss_seidel_iteration(psi)
369
370
371     ### End Method 2
372
373
374     print(np.amax(omega))
375
376     ### Method 1 (Fast):
377     # u_delta_T.fill(0)
378     # v_delta_T.fill(0)
379
380     # u_neg_ind = np.nonzero(u < 0)
381     # u_pos_ind = np.nonzero(u > 0)
382     # v_neg_ind = np.nonzero(v < 0)
383     # v_pos_ind = np.nonzero(v > 0)
384
385     # u_neg_ind_right = u_neg_ind[:]
386     # u_neg_ind_right[:,1][:] += 1           # Should this be 0?
387
388     # u_pos_ind_left = u_pos_ind[:]
389     # u_pos_ind_left[:,1][:] += -1          # Should this be 0?
390
391
392     # v_neg_ind_up = v_neg_ind[:]
393     # v_neg_ind_up[:,0][:] += 1             # Should this be 1?
394
395     # v_pos_ind_down = v_pos_ind[:]
396     # v_pos_ind_down[:,0][:] += -1         # Should this be 1?
397
398
399     # delta_u_omega[u_neg_ind] = u[u_neg_ind_right] * omega[u_neg_ind_right] - u[

```

```

400     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]
401
402     # delta_v_omega[v_neg_ind] = v[v_neg_ind_up] * omega[v_neg_ind_up] - v[v_neg_ind] *
      omega[v_neg_ind]
403     # delta_v_omega[v_pos_ind] = v[v_pos_ind] * omega[v_pos_ind] - v[v_pos_ind_down] *
      omega[v_pos_ind_down]
404
405     # vorticity_laplacian[CURRENT] = (omega[UP] + omega[DOWN] + omega[LEFT] + omega[
      RIGHT] - 4 * omega[CURRENT]) / (h * h)
406
407     # omega[CURRENT] += dt * (-delta_u_omega[CURRENT] - delta_v_omega[CURRENT]) / h + nu
      * vorticity_laplacian[CURRENT]
408
409
410
411
412
413     # psi = gauss_seidel_iteration(psi)
414
415
416
417
418
419     # u_delta_T[u_neg_ind] = u[u_neg_ind] * (temps_prev[u_neg_ind_right] - temps_prev[
      u_neg_ind])
420     # u_delta_T[u_pos_ind] = u[u_pos_ind] * (temps_prev[u_pos_ind] - temps_prev[
      u_pos_ind_left])
421
422     # v_delta_T[v_neg_ind] = v[v_neg_ind] * (temps_prev[v_neg_ind_up] - temps_prev[
      v_neg_ind])
423     # v_delta_T[v_pos_ind] = v[v_pos_ind] * (temps_prev[v_pos_ind] - temps_prev[
      v_pos_ind_down])
424

```

```

425 # temps_laplacian[CURRENT] = (temps_prev[UP] + temps_prev[DOWN] + temps_prev[LEFT] +
    temps_prev[RIGHT] - 4 * temps_prev[CURRENT]) / (h * h)
426
427 # temps = temps_prev + dt * ((-u_delta_T - v_delta_T) / h + alpha * temps_laplacian)
428
429 # temps[solid_rows, solid_cols] = T_surface
430 # temps[:, 0] = T_boundary
431 # temps[:, (height - 1)] = T_boundary
432
433
434 ### End Method 1
435
436
437
438 # Outflow Boundary Conditions:
439 psi[width - 1, :] = 2 * psi[width - 2, :] - psi[width - 3, :]
440 omega[width - 1, :] = omega[width - 2, :]
441
442
443
444 # if ((n + 1) % 10 == 0):
445 #     print("Time Step: " + str(n) + " of " + str(num_time_steps))
446
447 print("Time Step: " + str(n + 1) + " of " + str(num_time_steps))
448
449
450
451 omega_history.append(omega.copy())
452 psi_history.append(psi.copy())
453 temps_history.append(temps.copy())
454
455
456 print("\n--- Time Steps Done ---")
457 print("--- %.6f seconds ---\n" % (time.time() - start_time))
458

```

```

459
460
461 def print_data_in_console():
462     """ Print the data in the console (readable format) """
463     print(np.rot90(psi))
464 # print_data_in_console()
465
466
467 #####
468 #   Graphs and Plots
469 #####
470 def delete_previous_images():
471     image_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project\\
images"
472
473     old_images = [old_img for old_img in os.listdir(image_folder) if old_img.endswith(".
png")]
474     for old_image in old_images:
475         os.remove(os.path.join(image_folder, old_image))
476 delete_previous_images()
477
478
479 fig = plt.figure(figsize=(10, 10.5))
480
481 for plot_index in range(num_time_steps):
482     if final_frame_only and (plot_index != num_time_steps - 1):
483         continue
484
485     figNum = plot_index
486
487     #####
488     #   Streamfunction Plot
489     #####
490     sub1 = plt.subplot(3, 1, 1, aspect = 'equal')
491     data_graphable = np.flipud(np.rot90(psi_history[plot_index]))

```

```

492
493 plt.title("Streamfunction")
494
495 num_streamlines = 31
496 max_streamline = np.max(data_graphable)
497 min_streamline = np.min(data_graphable)
498 contours_before = np.linspace(min_streamline, max_streamline, num=(num_streamlines +
499 3))
500 contours = contours_before[(contours_before != 0) & (contours_before !=
501 min_streamline) & (contours_before != max_streamline)]
502
503
504 plt.contour(data_graphable, levels = contours, colors = 'black', linestyles = 'solid
505 ')
506
507
508
509 plt.style.use('grayscale')
510 plt.xticks(np.arange(0, width + 1, 50))
511 plt.yticks(np.arange(0, height + 1, 20))
512 plt.tick_params(top=True, right=True)
513
514
515 plt.pcolor(data_graphable)
516
517
518 # Color bar:
519 divider1 = make_axes_locatable(sub1)
520 cax1 = divider1.append_axes('right', size = '3%', pad = 0.3)
521 im = sub1.imshow(data_graphable, origin = 'lower', aspect = 'equal', interpolation =
522 'none')
523 fig.colorbar(im, cax = cax1, orientation = 'vertical')
524
525
526 #####
527 # Vorticity Plot
528 #####

```



```

523 sub2 = plt.subplot(3, 1, 2, aspect = 'equal')
524 data_graphable = np.flipud(np.rot90(omega_history[plot_index]))
525
526 plt.title("Vorticity")
527 plt.axis("off")
528
529 plt.pcolor(data_graphable)
530
531 # Color bar:
532 divider2 = make_axes_locatable(sub2)
533 cax2 = divider2.append_axes('right', size = '3%', pad = 0.3)
534 im = sub2.imshow(data_graphable, origin = 'lower', aspect = 'equal', interpolation =
535 'none')
536 fig.colorbar(im, cax = cax2, orientation = 'vertical')
537
538
539 #####
540 # Temperature Plot
541 #####
542 sub3 = fig.add_subplot(3, 1, 3, aspect = 'equal')
543 data_graphable = np.flipud(np.rot90(temps_history[plot_index]))
544 plt.title("Temperature")
545
546
547 plt.style.use('classic')
548 plt.axis("off")
549
550 # Color bar:
551 divider3 = make_axes_locatable(sub3)
552 cax3 = divider3.append_axes('right', size = '3%', pad = 0.3)
553 im = sub3.imshow(data_graphable, origin = 'lower', aspect = 'equal', interpolation =
554 'none')
555 cbar = fig.colorbar(im, cax=cax3, orientation = 'vertical')
556 cbar.set_label("Temperature (\N{DEGREE SIGN}C)")

```

```

556
557
558
559 #####
560 #   Time Stamp
561 #####
562 realtime = plot_index * dt
563 realtime_str = "Time: " + "{:.10f}".format(realtime) + " s"
564 fig.suptitle(realtime_str, y = 0.07)
565
566
567
568 #####
569 #   Save Frame
570 #####
571 plt.savefig(fileName + "/images/" + fileName + "-Figure-" + str(figNum + 1) + ".png"
572 )
573 plt.clf()
574
575 print("Frame: " + str(figNum + 1) + " of " + str(num_time_steps))
576
577 print("\n--- Figures Done ---")
578 print("--- %.6f seconds ---" % (time.time() - start_time))
579
580 #####
581 #   Generate Video
582 #####
583 def atoi(text):
584     # https://stackoverflow.com/questions/5967500/how-to-correctly-sort-a-string-with-a-number-inside
585     return int(text) if text.isdigit() else text
586
587 def natural_keys(text):
588     # https://stackoverflow.com/questions/5967500/how-to-correctly-sort-a-string-with-a

```

```

number-inside
589     return [ atoi(c) for c in re.split(r'(\d+)', text) ]
590
591 def generate_video():
592     image_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project\\
images"
593     output_folder = "C:\\Users\\samda\\Documents\\GitHub\\Heat-Transfer\\Final-Project\\"
"
594     video_name = output_folder + "final-project.mp4"
595     fps = 5
596
597     images = [img for img in os.listdir(image_folder) if img.endswith(".png")]
598     images.sort(key=natural_keys)
599     frame = cv2.imread(os.path.join(image_folder, images[0]))
600     height, width, layers = frame.shape
601
602     fourcc = cv2.VideoWriter_fourcc(*'mp4v')
603     video = cv2.VideoWriter(video_name, fourcc, fps, (width, height))
604
605     for image in images:
606         video.write(cv2.imread(os.path.join(image_folder, image)))
607
608     cv2.destroyAllWindows()
609     video.release()
610 if generate_video:
611     generate_video()
612
613 print("\n--- Video Done ---")
614 print("--- %.6f seconds ---" % (time.time() - start_time))

```