

# Листинг программы расчета аэродинамических характеристик

Программа для определения коэффициента подъемной силы летательного аппарата

## Аннотация

Программа реализована на языке Python с использованием библиотеки AeroBDSM для расчета аэродинамических характеристик. Выполняет расчет коэффициента подъемной силы с учетом интерференции между корпусом и несущими поверхностями, скоса потока, а также влияния числа Маха и угла атаки.

## Основной код программы

```
1 import AeroBDSM
2 import math
3 import csv
4 import numpy as np
5 from tqdm import tqdm
6 import warnings
7 warnings.filterwarnings("ignore", category=RuntimeWarning)
8 class ProbabilityIntegral:
9     def __init__(self):
10         # Полная таблица значений (x) с исправленными опечатками
11         self.table = {
12             # Первая часть таблицы (0.00 - 1.20)
13             0.00: 0.0000, 0.01: 0.0080, 0.02: 0.0160, 0.03: 0.0239, 0.04: 0.0319,
14             0.05: 0.0399, 0.06: 0.0478, 0.07: 0.0558, 0.08: 0.0638, 0.09: 0.0717,
15             0.10: 0.0797, 0.11: 0.0876, 0.12: 0.0955, 0.13: 0.1034, 0.14: 0.1113,
16             0.15: 0.1192, 0.16: 0.1271, 0.17: 0.1350, 0.18: 0.1428, 0.19: 0.1507,
17             0.20: 0.1585, 0.21: 0.1663, 0.22: 0.1741, 0.23: 0.1819, 0.24: 0.1897,
18             0.25: 0.1974, 0.26: 0.2051, 0.27: 0.2128, 0.28: 0.2205, 0.29: 0.2282,
19             0.30: 0.2358, 0.31: 0.2434, 0.32: 0.2510, 0.33: 0.2586, 0.34: 0.2661,
20             0.35: 0.2737, 0.36: 0.2812, 0.37: 0.2886, 0.38: 0.2961, 0.39: 0.3035,
21             0.40: 0.3108, 0.41: 0.3182, 0.42: 0.3255, 0.43: 0.3328, 0.44: 0.3401,
22             0.45: 0.3473, 0.46: 0.3545, 0.47: 0.3616, 0.48: 0.3688, 0.49: 0.3759,
23             0.50: 0.3829, 0.51: 0.3899, 0.52: 0.3969, 0.53: 0.4039, 0.54: 0.4108,
24             0.55: 0.4177, 0.56: 0.4245, 0.57: 0.4313, 0.58: 0.4381, 0.59: 0.4448,
25             0.60: 0.4515, 0.61: 0.4581, 0.62: 0.4647, 0.63: 0.4713, 0.64: 0.4778,
26             0.65: 0.4843, 0.66: 0.4907, 0.67: 0.4971, 0.68: 0.5035, 0.69: 0.5098,
27             0.70: 0.5161, 0.71: 0.5223, 0.72: 0.5285, 0.73: 0.5346, 0.74: 0.5407,
28             0.75: 0.5467, 0.76: 0.5527, 0.77: 0.5587, 0.78: 0.5646, 0.79: 0.5705,
29             0.80: 0.5763, 0.81: 0.5821, 0.82: 0.5878, 0.83: 0.5935, 0.84: 0.5991,
30             0.85: 0.6047, 0.86: 0.6102, 0.87: 0.6157, 0.88: 0.6211, 0.89: 0.6265,
31             0.90: 0.6319, 0.91: 0.6372, 0.92: 0.6424, 0.93: 0.6476, 0.94: 0.6528,
32             0.95: 0.6579, 0.96: 0.6629, 0.97: 0.6680, 0.98: 0.6729, 0.99: 0.6778,
33             1.00: 0.6827, 1.01: 0.6875, 1.02: 0.6923, 1.03: 0.6970, 1.04: 0.7017,
34             1.05: 0.7063, 1.06: 0.7109, 1.07: 0.7154, 1.08: 0.7199, 1.09: 0.7243,
35             1.10: 0.7287, 1.11: 0.7330, 1.12: 0.7373, 1.13: 0.7415, 1.14: 0.7457,
36             1.15: 0.7499, 1.16: 0.7540, 1.17: 0.7580, 1.18: 0.7620, 1.19: 0.7660,
37
38             # Вторая часть таблицы (1.20 - 4.90)
39             1.20: 0.7699, 1.21: 0.7737, 1.22: 0.7775, 1.23: 0.7813, 1.24: 0.7850,
40             1.25: 0.7887, 1.26: 0.7923, 1.27: 0.7959, 1.28: 0.7995, 1.29: 0.8029,
41             1.30: 0.8064, 1.31: 0.8098, 1.32: 0.8132, 1.33: 0.8165, 1.34: 0.8198,
42             1.35: 0.8230, 1.36: 0.8262, 1.37: 0.8293, 1.38: 0.8324, 1.39: 0.8355,
43             1.40: 0.8385, 1.41: 0.8415, 1.42: 0.8444, 1.43: 0.8473, 1.44: 0.8501,
44             1.45: 0.8529, 1.46: 0.8557, 1.47: 0.8584, 1.48: 0.8611, 1.49: 0.8638,
45             1.50: 0.8664, 1.51: 0.8690, 1.52: 0.8715, 1.53: 0.8740, 1.54: 0.8764,
46             1.55: 0.8789, 1.56: 0.8812, 1.57: 0.8836, 1.58: 0.8859, 1.59: 0.8882,
47             1.60: 0.8904, 1.61: 0.8926, 1.62: 0.8948, 1.63: 0.8969, 1.64: 0.8990,
48             1.65: 0.9011, 1.66: 0.9031, 1.67: 0.9051, 1.68: 0.9070, 1.69: 0.9090,
49             1.70: 0.9109, 1.71: 0.9127, 1.72: 0.9146, 1.73: 0.9164, 1.74: 0.9181,
50             1.75: 0.9199, 1.76: 0.9216, 1.77: 0.9233, 1.78: 0.9249, 1.79: 0.9265,
51             1.80: 0.9281, 1.81: 0.9297, 1.82: 0.9312, 1.83: 0.9328, 1.84: 0.9342,
52             1.85: 0.9357, 1.86: 0.9371, 1.87: 0.9385, 1.88: 0.9399, 1.89: 0.9412,
53             1.90: 0.9426, 1.91: 0.9439, 1.92: 0.9451, 1.93: 0.9464, 1.94: 0.9476,
54             1.95: 0.9488, 1.96: 0.9500, 1.97: 0.9512, 1.98: 0.9523, 1.99: 0.9534,
55             2.00: 0.9545, 2.05: 0.9596, 2.10: 0.9643, 2.15: 0.9684, 2.20: 0.9722,
56             2.25: 0.9756, 2.30: 0.9786, 2.35: 0.9812, 2.40: 0.9836, 2.45: 0.9857,
57             2.50: 0.9876, 2.55: 0.9892, 2.60: 0.9907, 2.65: 0.9920, 2.70: 0.9931,
58             2.75: 0.9940, 2.80: 0.9949, 2.85: 0.9956, 2.90: 0.9963, 2.95: 0.9968,
```

```

59         3.00: 0.99730, 3.10: 0.99806, 3.20: 0.99863, 3.30: 0.99903, 3.40: 0.99933,
60         3.50: 0.99953, 3.60: 0.99968, 3.70: 0.99978, 3.80: 0.99986, 3.90: 0.99990,
61         4.00: 0.99994, 4.417: 0.99999, 4.852: 0.999999, 5.327: 0.9999999
62     }
63
64     # Исправления опечаток из таблицы
65     self.table[0.46] = 0.3545 # Исправлено с 0.3845
66     self.table[0.49] = 0.3759 # Исправлено с 0.3769
67     self.table[0.57] = 0.4313 # Исправлено с 0.4813
68     self.table[0.68] = 0.5035 # Исправлено с 0.5095
69
70     def get_value(self, x):
71         """Получить значение (x) с линейной интерполяцией"""
72         # Если значение есть в таблице - возвращаем его
73         if x in self.table:
74             return self.table[x]
75
76         # Получаем отсортированные ключи
77         x_values = sorted(self.table.keys())
78
79         # Если x меньше минимального значения
80         if x < x_values[0]:
81             return 0.0
82
83         # Если x больше максимального значения
84         if x > x_values[-1]:
85             return 1.0 # (x) стремится к 1 при x
86
87         # Линейная интерполяция между ближайшими значениями
88         for i in range(len(x_values) - 1):
89             if x_values[i] <= x <= x_values[i + 1]:
90                 x1, x2 = x_values[i], x_values[i + 1]
91                 y1, y2 = self.table[x1], self.table[x2]
92                 return y1 + (y2 - y1) * (x - x1) / (x2 - x1)
93
94         return self.table[x_values[-1]]
95
96     phi = ProbabilityIntegral()
97
98
99
100     # Глобальные переменные
101     l_f = 3.906 # длина фюзеляжа м
102     # M = 0.0
103     pi = math.pi
104     D = 0.178 # диаметр миделя, м aka диаметр миделевого сечения корпуса, м
105     D_I = 0.178 # диаметр корпуса в области передних консолей, м
106     D_II = 0.178 # диаметр корпуса в области задних консолей, м
107     D_bar = D / l_f # относительный диаметр корпуса
108
109     l_nos = 0.47 # длина носа м
110     l_korm = 0.375 # длина кормы, м
111     S_m = (pi * pow(D, 2.0)) / 4.0 # площадь миделя, м^2
112     l_I = 0.484 # размах крыла м
113     l_II = 0.5808 # размах руля м
114
115     # Коэффициенты интерференции
116     k_aa_t = pow(1 + 0.41 * D_bar, 2)
117     K_aa_t = pow(1 + D_bar, 2)
118     eta_k = 1 # обратное сужение
119
120     b_b = 0.2 # бортовая хорда
121
122     L_xv = 1.701 # длина хвостовой части (от конца бортовой хорды до конца короче)
123
124     c = 0.004 # толщина профиля
125     x_b = 1 # что это ? стр.162
126     L_1 = x_b + (b_b / 2)
127     nu = 15.1 * 1e-6 # кин. кэф. вязкости воздуха стр.162
128     L1_bar = L_1 / D
129     x_M = 1 # откуда??
130     y_v = 1.0
131
132     # для рулей:
133     eta_k_rl = 1
134     b_b_rl = 0.316

```

```

135 L_xv_rl = 1.701
136 L_1_rl = x_b + ( b_b_rl/ 2)
137 L1_bar_rl=L_1_rl/D
138 lambda_rl = 2.805
139 chi_05_rl = 0.420
140 zeta_rl = 0.0348
141 #////////////////////////////////////
142 #для get_psi_eps:
143 M_values = np.linspace(0.1,4.1,36)
144
145 alpha_p_values = np.linspace(0,0.436332,10) #от 0 до +25 градусов
146
147 phi_alpha_values = np.linspace(0, 0.436332, 10) #от 0 до +25 градусов
148
149 psi_I_values = np.linspace(0, 0.436332, 10) #от 0 до +25 градусов
150
151 psi_II_values = np.linspace(0, 0.436332, 10) #от 0 до +25 градусов
152
153 x_zI_II = 1
154 d_II = D
155 l_1c_II = 0.231
156 zeta_II = zeta_rl
157 b_b_II = 0.316
158 chi_0_II =0.420
159
160 #////////////////////////////////////
161
162
163 lambda_kr = 2.49 # удлинение несущей поверхности
164 chi_05_kr = 0.510 # угол стреловидности по линии середин хорд, рад
165 bar_c_kr = 0.224 # относительная толщина профиля
166 zeta_kr = 0.2 # обратное сужение несущей поверхности
167 # Дополнительные параметры
168 b_kr = 0.1 # Хорда крыла, м
169 b_op = 0.025 # Хорда оперения, м
170 l_raszmah_kr = 0.498 # Размах крыла, м
171 l_raszmah_rl = 0.651 # Размах оперения, м
172
173 # //////////////////////////////////
174 # расчет c^\alpha_y
175
176 S_f = 2.1842
177
178 S_kr = 0.0996
179
180 S_rl = 0.15107
181
182 S = S_f + S_kr + S_rl
183
184 S_f_bar = S_f / S # Относительная площадь фюзеляжа
185 S_1_bar = S_kr / S # Относительная площадь оперения
186 S_2_bar = S_rl / S # Относительная площадь крыла
187
188
189
190 # //////////////////////////////////
191
192 # Площади поверхностей
193 S_Kons = 4 * 0.03375 * 0.008 # Площадь консолей крыла, м
194 S_op = 2 * 0.033 * 0.025 # Площадь оперения, м
195
196 L_A = 1
197 b_A = 1.5
198
199
200 # # Расчетные параметры крыльевых поверхностей
201 # l_raszmah_kons = l_raszmah - D # Размах консоли крыла
202 # lamb = pow(l_raszmah_kons, 2.0) / S_Kons # Удлинение крыла
203 # lamb_op = pow(l_raszmah_rl, 2.0) / (S_op + pow(b_kr, 2.0)) # Удлинение оперения
204
205 # Итоговые параметры
206 S_f = 2.0 * pi * D * (l_f - D / 2.0) + 2.0 * pi * pow(D / 2.0, 2.0) # Площадь поверхности фюз
    еляжа, м
207
208 def save_data_to_csv(x, y, filename):
209     """Сохраняет данные в CSV файл"""

```

```

210     with open(filename, 'w', newline='') as file:
211         writer = csv.writer(file)
212         writer.writerow(['x', 'y'])
213         for i in range(len(x)):
214             writer.writerow([x[i], y[i]])
215
216 def main():
217     angles_deg = [-10, -5, 0, 5, 10]
218     lambda_Nos = l_nos / D
219     lambda_Cil = l_f / D
220
221     # Первый расчет - носовая часть
222     with open('all_angles_data.csv', 'w', newline='') as file1:
223         writer1 = csv.writer(file1)
224         header1 = ['Mach'] + [f'alpha_{angle}' for angle in angles_deg]
225         writer1.writerow(header1)
226
227         M = 0.5
228         while M <= 4.1:
229             result = AeroBDSM.get_c_y_alpha_NosCil_Con(M, lambda_Nos, lambda_Cil)
230             if result.ErrorCode == 0:
231                 c_y_alpha = result.Value
232                 row = [M]
233                 for angle_deg in angles_deg:
234                     angle_rad = angle_deg * math.pi / 180.0
235                     c_y = c_y_alpha * angle_rad
236                     row.append(c_y)
237                 writer1.writerow(row)
238                 M += 0.1
239
240     # Второй расчет - крыло изолированное
241     with open('krylo_isP.csv', 'w', newline='') as file2:
242         writer2 = csv.writer(file2)
243         header2 = ['Mach'] + [f'alpha_{angle}' for angle in angles_deg]
244         writer2.writerow(header2)
245
246         M = 0.5
247         while M <= 4.1:
248             result2 = AeroBDSM.get_c_y_alpha_IsP(M, lambda_kr, bar_c_kr, chi_05_kr, zeta_kr)
249             if result2.ErrorCode == 0:
250                 c_y_alpha = result2.Value
251                 row = [M]
252                 for angle_deg in angles_deg:
253                     angle_rad = angle_deg * math.pi / 180.0
254                     c_y = c_y_alpha * angle_rad
255                     row.append(c_y)
256                 writer2.writerow(row)
257                 M += 0.1
258
259     with open('Kappa_aa_kr.csv', 'w', newline='') as file4:
260         writer4 = csv.writer(file4)
261         header4 = ['Mach', 'K_aa'] + [f'alpha_{angle}' for angle in angles_deg]
262         writer4.writerow(header4)
263
264         M = 0.5
265         while M <= 4.1:
266             V = 342 * M
267
268             # Исправление для избежания math domain error
269             if M < 1:
270                 sqrt_term = 0 # Для M < 1 используем 0
271             else:
272                 sqrt_term = math.sqrt(M**2 - 1)
273
274             b_b_bar = b_b / ((math.pi/2)*D* sqrt_term) if sqrt_term > 0 else 0
275             if b_b_bar < 0 or M == 2.0000000000000004 :
276                 print(f'b_b_bar = {b_b_bar:.3f}')
277
278             L_xv_bar = (L_xv / ((math.pi/2)*D* sqrt_term)) if sqrt_term > 0 else 0
279             if L_xv_bar < 0 or M == 2.0000000000000004 :
280                 print(f'L_xv_bar = {L_xv_bar:.3f}')
281
282             # Проверка деления на ноль для F_L_xv
283             if b_b_bar > 0 and c > 0:
284                 sqrt_2c = math.sqrt(2*c)

```

```

285         F_L_xv = 1 - (math.sqrt(math.pi) / (2 * b_b_bar * sqrt_2c)) *
                (phi.get_value((b_b_bar + L_xv_bar) * sqrt_2c) - phi.get_value(L_xv_bar *
                sqrt_2c))
286     else:
287         F_L_xv = 1.0
288
289     if F_L_xv < 0 or M == 2.00000000000000042:
290         print(f'F_L_xv = {F_L_xv:.3f}')
291
292     delta_zvz_bar_kr = (0.093 / ((V * L_1) / nu)**(1/5)) * (L_1 / D) * (1 + 0.4*M +
293     0.147 * M**2 - 0.006 * M**3)
294     if delta_zvz_bar_kr < 0 or M == 2.0000000000000004:
295         print(f'delta_zvz_bar_kr = {delta_zvz_bar_kr:.3f}')
296
297     x_ps = (1 - ((2 * D_bar)/(1 - D_bar**2)) * delta_zvz_bar_kr) * (1 - ((D_bar *
298     (eta_k-1))/(1 - D_bar) * (eta_k + 1)) * delta_zvz_bar_kr)
299     if x_ps < 0 or M == 2.0000000000000004 :
300         print(f'x_ps = {x_ps:.3f}')
301
302     x_nos = 0.6 + 0.4 * (1 - math.exp(-0.5 * L1_bar))
303     if x_nos < 0 or M == 2.0000000000000004:
304         print(f'x_nos = {x_nos:.3f}')
305
306     k_aa_zvz = K_aa_t * ((1 + 3*D_bar - (1 / eta_k) * D_bar * (1 - D_bar)) / (1 +
307     D_bar)**2)
308     if k_aa_zvz < 0 or M == 2.00000000000000042:
309         print(f'k_aa_zvz = {k_aa_zvz:.3f}')
310
311     K_aa_zvz = 1 + 3 * D_bar - ((D_bar * (1 - D_bar)) / eta_k)
312     if K_aa_zvz < 0 or M == 2.0000000000000004:
313         print(f'K_aa_zvz = {K_aa_zvz:.3f}')
314
315     k_aa = K_aa_zvz * x_ps * x_M * x_nos
316     if M >= 1 :
317         K_aa = (k_aa_zvz + (K_aa_zvz - k_aa_zvz) * F_L_xv) * x_ps * x_M * x_nos
318     else:
319         K_aa = K_aa_zvz * F_L_xv * x_ps * x_M * x_nos
320
321     row = [M, K_aa]
322     for angle_deg in angles_deg:
323         angle_rad = angle_deg * math.pi / 180.0
324         row.append(K_aa)
325
326     writer4.writerow(row)
327
328     M += 0.1
329
330 # пятый расчет крыло + good интерфер-я
331 with open('krylo_isP_Intrf.csv', 'w', newline='') as file5:
332     writer5 = csv.writer(file5)
333     header5 = ['Mach'] + [f'alpha_{angle}' for angle in angles_deg]
334     writer5.writerow(header5)
335
336     M = 0.5
337     while M <= 4.1:
338         result5 = AeroBDSM.get_c_y_alpha_IsP(M, lambda_kr, bar_c_kr, chi_05_kr, zeta_kr)
339         if result5.ErrorCode == 0:
340             c_y_alpha = result5.Value * K_aa
341             row = [M]
342             for angle_deg in angles_deg:
343                 angle_rad = angle_deg * math.pi / 180.0
344                 c_y = c_y_alpha * angle_rad
345                 row.append(c_y)
346             writer5.writerow(row)
347         M += 0.1
348
349 # шестой расчет - руль изолированный
350 with open('rul_isP.csv', 'w', newline='') as file6:
351     writer6 = csv.writer(file6)
352     header6 = ['Mach'] + [f'alpha_{angle}' for angle in angles_deg]
353     writer6.writerow(header6)
354
355     M = 0.5
356     while M <= 4.1:
357         result6 = AeroBDSM.get_c_y_alpha_IsP(M, lambda_rl, bar_c_kr, chi_05_rl, zeta_rl)
358         if result6.ErrorCode == 0:

```

```

356         c_y_alpha = result6.Value
357         row = [M]
358         for angle_deg in angles_deg:
359             angle_rad = angle_deg * math.pi / 180.0
360             c_y = c_y_alpha * angle_rad
361             row.append(c_y)
362         writer6.writerow(row)
363     M += 0.1
364 print(60*'=' )
365 with open('Kappa_aa_rl.csv', 'w', newline='') as file7:
366     writer7 = csv.writer(file7)
367     header7 = ['Mach', 'K_aa'] + [f'alpha_{angle}' for angle in angles_deg]
368     writer7.writerow(header7)
369
370 M = 0.5
371 while M <= 4.1:
372     V = 342 * M
373
374     # Исправление для избежания math domain error
375     if M < 1:
376         sqrt_term = 0 # Для M < 1 используем 0
377     else:
378         sqrt_term = math.sqrt(M**2 - 1)
379
380     b_b_bar_rl = b_b / ((math.pi/2)*D* sqrt_term) if sqrt_term > 0 else 0
381     if b_b_bar_rl < 0 or M == 2.0000000000000004 :
382         print(f'b_b_bar_rl = {b_b_bar_rl:.3f}')
383
384     L_xv_bar_rl = (L_xv_rl / ((math.pi/2)*D* sqrt_term)) if sqrt_term > 0 else 0
385     if L_xv_bar_rl < 0 or M == 2.0000000000000004 :
386         print(f'L_xv_bar_rl = {L_xv_bar_rl:.3f}')
387
388     # Проверка деления на ноль для F_L_xv
389     if b_b_bar_rl > 0 and c > 0:
390         sqrt_2c = math.sqrt(2*c)
391         F_L_xv_rl = 1 - (math.sqrt(math.pi) / (2 * b_b_bar_rl * sqrt_2c)) *
392             (phi.get_value((b_b_bar_rl + L_xv_bar_rl)* sqrt_2c) -
393              phi.get_value(L_xv_bar_rl * sqrt_2c))
394     else:
395         F_L_xv_rl = 1.0
396
397     if F_L_xv_rl < 0 or M == 2.00000000000000042:
398         print(f'F_L_xv_rl = {F_L_xv_rl:.3f}')
399
400     delta_zvz_bar_rl = (0.093 / ((V * L1_rl) / nu)**(1/5)) * (L1_rl / D) * (1 +
401         0.4*M + 0.147 * M**2 - 0.006 * M**3)
402     if delta_zvz_bar_rl < 0 or M == 2.0000000000000004:
403         print(f'delta_zvz_bar_rl = {delta_zvz_bar_rl:.3f}')
404
405     x_ps_rl = (1 - ((2 * D_bar)/(1 - D_bar**2)) * delta_zvz_bar_rl) * (1 - ((D_bar *
406         (eta_k_rl - 1))/(1 - D_bar) * (eta_k_rl + 1)) * delta_zvz_bar_rl)
407     if x_ps_rl < 0 or M == 2.0000000000000004 :
408         print(f'x_ps_rl = {x_ps_rl:.3f}')
409
410     x_nos_rl = 0.6 + 0.4 * (1 - math.exp(-0.5 * L1_bar_rl))
411     if x_nos_rl < 0 or M == 2.0000000000000004:
412         print(f'x_nos_rl = {x_nos_rl:.3f}')
413
414     k_aa_zvz_rl = K_aa_t * ((1 + 3*D_bar - (1 / eta_k_rl) * D_bar * (1 - D_bar)) / (1
415         + D_bar)**2)
416     if k_aa_zvz_rl < 0 or M == 2.00000000000000042:
417         print(f'k_aa_zvz_rl = {k_aa_zvz_rl:.3f}')
418
419     K_aa_zvz_rl = 1 + 3 * D_bar - ((D_bar * (1 - D_bar)) / eta_k_rl)
420     if K_aa_zvz_rl < 0 or M == 2.0000000000000004:
421         print(f'K_aa_zvz_rl = {K_aa_zvz_rl:.3f}')
422
423     if M >= 1 :
424         K_aa_rl = (k_aa_zvz_rl + (K_aa_zvz_rl - k_aa_zvz_rl) * F_L_xv_rl) * x_ps_rl *
425             x_M * x_nos_rl
426     else:
427         K_aa_rl = K_aa_zvz_rl * F_L_xv_rl * x_ps_rl * x_M * x_nos_rl
428
429     row = [M, K_aa]
430     for angle_deg in angles_deg:
431         angle_rad = angle_deg * math.pi / 180.0

```

```

426         row.append(K_aa_rl)
427
428     writer7.writerow(row)
429
430     M += 0.1
431
432     #Учет схода потока
433 with open('z_b_v.csv', 'w', newline='') as file8:
434     writer8 = csv.writer(file8)
435     header8 = ['Mach', 'z_b']
436     writer8.writerow(header8)
437     M = 0.5
438     while M <= 4.1:
439         result8 = AeroBDSM.get_bar_z_v(M, lambda_kr, chi_05_kr, zeta_kr)
440         if result8.ErrorCode == 0:
441             z_b = result8.Value
442             row = [M, z_b]
443             writer8.writerow(row)
444         M += 0.1
445
446
447 # Учет схода потока - расчет i_v
448 with open('i_v.csv', 'w', newline='') as file9:
449     writer9 = csv.writer(file9)
450     header9 = ['Mach', 'i_v']
451     writer9.writerow(header9)
452     M = 0.5
453     while M <= 4.1:
454         # Сначала получаем z_b для текущего M
455         result8 = AeroBDSM.get_bar_z_v(M, lambda_kr, chi_05_kr, zeta_kr)
456         if result8.ErrorCode == 0:
457             z_b = result8.Value
458             # Затем рассчитываем i_v с полученным z_b
459             result9 = AeroBDSM.get_i_v(zeta_rl, D, l_raszmah_rl, y_v, z_b)
460             if result9.ErrorCode == 0:
461                 i_v = result9.Value
462                 row = [M, i_v]
463                 writer9.writerow(row)
464             M += 0.1
465
466
467 with open('psi_eps.csv', 'w', newline='') as file10:
468     writer10 = csv.writer(file10)
469     header10 = ['Mach', 'alpha_p', 'phi_alpha', 'psi_I', 'psi_II', 'z_v', 'psi_eps']
470     writer10.writerow(header10)
471     total_iterations = len(psi_II_values) * len(psi_I_values) * len(phi_alpha_values) *
472         len(alpha_p_values) * len(M_values)
473     with tqdm(total=total_iterations, desc="Расчет psi_eps") as pbar:
474         for psi_II in psi_II_values:
475             for psi_I in psi_I_values:
476                 for phi_alpha in phi_alpha_values:
477                     for alpha_p in alpha_p_values:
478                         for M in M_values:
479                             result8 = AeroBDSM.get_bar_z_v(M, lambda_kr, chi_05_kr,
480                                 zeta_kr)
481                             if result8.ErrorCode == 0:
482                                 z_v = result8.Value
483                                 result10 = AeroBDSM.get_psi_eps(M, alpha_p, phi_alpha,
484                                     psi_I, psi_II, z_v, y_v, x_zI_II, d_II,
485                                     l_1c_II, zeta_II, b_b_II, chi_0_II)
486                                 if result10.ErrorCode == 0:
487                                     psi_eps = result10.Value
488                                     row = [M, alpha_p, phi_alpha, psi_I, psi_II, z_v,
489                                         psi_eps]
490                                     writer10.writerow(row)
491                                 pbar.update(1)
492
493
494 with open('eps_alpha_sr.csv', 'w', newline='') as file11:
495     writer11 = csv.writer(file11)
496     header11 = ['eps_alpha_sr', 'Mach', 'alpha_p', 'phi_alpha', 'psi_I', 'psi_II', 'z_v',
497         'psi_eps', 'i_v', 'k_aa', 'K_aa_rl']
498     writer11.writerow(header11)
499     total_iterations = len(psi_II_values) * len(psi_I_values) * len(phi_alpha_values) *
500         len(alpha_p_values) * len(M_values)
501     with tqdm(total=total_iterations, desc="eps_alpha_sr") as pbar:
502         for psi_II in psi_II_values:
503             for psi_I in psi_I_values:

```

```

497     for phi_alpha in phi_alpha_values:
498         for alpha_p in alpha_p_values:
499             for M in M_values:
500                 result8 = AeroBDSM.get_bar_z_v(M, lambda_kr, chi_05_kr,
501                     zeta_kr)
502                 if result8.ErrorCode == 0:
503                     z_v = result8.Value
504                     result10 = AeroBDSM.get_psi_eps(M, alpha_p, phi_alpha,
505                         psi_I, psi_II, z_v, y_v, x_zI_II, d_II,
506                         l_1c_II, zeta_II, b_b_II, chi_0_II)
507
508                 if result10.ErrorCode == 0:
509
510                     result9 = AeroBDSM.get_i_v(zeta_rl, D, l_raszmah_rl,
511                         y_v, z_v)
512                     i_v = result9.Value
513
514                     psi_eps = result10.Value
515
516                     result2 = AeroBDSM.get_c_y_alpha_IsP(M, lambda_kr,
517                         bar_c_kr, chi_05_kr, zeta_kr)
518
519                     c_y_alpha_iz_kr = result2.Value
520
521                     V = 342 * M
522                     # Исправление для избежания math domain error
523                     if M < 1:
524                         sqrt_term = 0 # Для M < 1 используем 0
525                     else:
526                         sqrt_term = math.sqrt(M**2 - 1)
527
528                     b_b_bar_rl = b_b / ((math.pi/2)*D* sqrt_term) if
529                         sqrt_term > 0 else 0
530
531                     L_xv_bar_rl = (L_xv_rl/ ((math.pi/2)*D* sqrt_term))
532                         if sqrt_term > 0 else 0
533
534                     # Проверка деления на ноль для F_L_xv
535                     if b_b_bar_rl > 0 and c > 0:
536                         sqrt_2c = math.sqrt(2*c)
537                         F_L_xv_rl = 1 - (math.sqrt(math.pi) / (2 *
538                             b_b_bar_rl * sqrt_2c)) *
539                             (phi.get_value((b_b_bar_rl + L_xv_bar)*
540                                 sqrt_2c) - phi.get_value(L_xv_bar * sqrt_2c))
541                     else:
542                         F_L_xv_rl = 1.0
543
544                     delta_zvz_bar_rl = (0.093 / ((V * L_1_rl) /
545                         nu)**(1/5)) * (L_1_rl / D) * (1 + 0.4*M + 0.147 *
546                         M**2 - 0.006 * M**3)
547
548                     x_ps_rl = (1 - ((2 * D_bar)/(1 - D_bar**2)) *
549                         delta_zvz_bar_rl) * (1 - ((D_bar * (eta_k-1))/(1
550                             - D_bar) * (eta_k_rl + 1)) * delta_zvz_bar_rl)
551
552                     x_nos_rl = 0.6 + 0.4 * (1 - math.exp(-0.5 *
553                         L1_bar_rl))
554
555                     k_aa_zvz_rl = K_aa_t * ((1 + 3*D_bar - (1 / eta_k) *
556                         D_bar * (1 - D_bar)) / (1 + D_bar)**2)
557
558                     K_aa_zvz_rl = 1 + 3 * D_bar - ((D_bar * (1 - D_bar))
559                         / eta_k_rl)
560
561                     if M >=1 :
562                         K_aa_rl = (k_aa_zvz_rl + (K_aa_zvz_rl -
563                             k_aa_zvz_rl) * F_L_xv_rl) * x_ps_rl * x_M *
564                             x_nos_rl
565                     else:
566                         K_aa_rl = K_aa_zvz_rl * F_L_xv_rl * x_ps_rl * x_M
567                             * x_nos_rl
568
569                     #////////////////////////////////////

```



```

555         if M < 1:
556             sqrt_term = 0 # Для M < 1 используем 0
557         else:
558             sqrt_term = math.sqrt(M**2 - 1)
559
560         b_b_bar = b_b / ((math.pi/2)*D* sqrt_term) if
                    sqrt_term > 0 else 0
561
562         L_xv_bar = (L_xv/ ((math.pi/2)*D* sqrt_term)) if
                    sqrt_term > 0 else 0
563
564         # Проверка деления на ноль для F_L_xv
565         if b_b_bar > 0 and c > 0:
566             sqrt_2c = math.sqrt(2*c)
567             F_L_xv = 1 - (math.sqrt(math.pi) / (2 * b_b_bar *
                    sqrt_2c)) * (phi.get_value((b_b_bar +
                    L_xv_bar)* sqrt_2c) - phi.get_value(L_xv_bar
                    * sqrt_2c))
568         else:
569             F_L_xv = 1.0
570
571         delta_zvz_bar_kr = (0.093 / ((V * L_1) / nu)**(1/5))
                    * (L_1 / D) * (1 + 0.4*M + 0.147 * M**2 - 0.006 *
                    M**3)
572
573         x_ps = (1 - ((2 * D_bar)/(1 - D_bar**2)) *
                    delta_zvz_bar_kr) * (1 - ((D_bar * (eta_k-1))/(1
                    - D_bar) * (eta_k + 1)) * delta_zvz_bar_kr)
574
575         x_nos = 0.6 + 0.4 * (1 - math.exp(-0.5 * L1_bar))
576
577         k_aa_zvz = K_aa_t * ((1 + 3*D_bar - (1 / eta_k) *
                    D_bar * (1 - D_bar)) / (1 + D_bar)**2)
578
579         K_aa_zvz = 1 + 3 * D_bar - ((D_bar * (1 - D_bar)) /
                    eta_k)
580
581         k_aa = K_aa_zvz * x_ps * x_M * x_nos
582
583         eps_alpha_sr = (57.3/(2*math.pi)) * (i_v/z_v) *
                    (l_rasmah_kr/l_rasmah_rl) *
                    (c_y_alpha_iz_kr/lambda_kr) * (k_aa/K_aa_rl) *
                    psi_eps
584         row = [eps_alpha_sr, M, alpha_p, phi_alpha, psi_I,
                    psi_II, z_v, psi_eps, i_v, k_aa, K_aa_rl]
585         writer11.writerow(row)
586         pbar.update(1)
587
588
589
590
591     with open('kappa_q_nos.csv', 'w', newline='') as file12:
592         writer12 = csv.writer(file12)
593         header12 = ['Mach', 'kappa_q']
594         writer12.writerow(header12)
595         M = 1
596         while M <= 4.1:
597             result12 = AeroBDSM.get_kappa_q_Nos_Con(M, lambda_Nos)
598             if result12.ErrorCode == 0:
599                 kappa_q = result12.Value
600                 row = [M, kappa_q]
601                 writer12.writerow(row)
602                 M += 0.1
603
604
605     with open('kappa_q.csv', 'w', newline='') as file13:
606         writer13 = csv.writer(file13)
607         header13 = ['Mach', 'kappa_q']
608         writer13.writerow(header13)
609         M = 0.5
610         while M <= 4.1:
611             result13 = AeroBDSM.get_kappa_q_IsP(M, L_A, b_A)
612             if result13.ErrorCode == 0:
613                 kappa_q = result13.Value
614                 row = [M, kappa_q]
615                 writer13.writerow(row)

```

```

616         M += 0.1
617
618     #расчет c_y\alpha_y
619     with open('c_y_alpha_sum.csv', 'w', newline='') as file14:
620         writer14 = csv.writer(file14)
621         header14 = ['Mach'] + [f'alpha_{angle}' for angle in angles_deg]
622         writer14.writerow(header14)
623
624     M = 0.5
625     while M <= 4.1:
626         # Получаем все необходимые значения для текущего M
627         result = AeroBDSM.get_c_y_alpha_NosCil_Con(M, lambda_Nos, lambda_Cil)
628         result2 = AeroBDSM.get_c_y_alpha_IsP(M, lambda_kr, bar_c_kr, chi_05_kr, zeta_kr)
629         result6 = AeroBDSM.get_c_y_alpha_IsP(M, lambda_rl, bar_c_kr, chi_05_rl, zeta_rl)
630         result12 = AeroBDSM.get_kappa_q_Nos_Con(M, lambda_Nos)
631         result13 = AeroBDSM.get_kappa_q_IsP(M, L_A, b_A)
632
633         # Проверяем все ошибки
634         if (result.ErrorCode == 0 and result2.ErrorCode == 0 and
635             result6.ErrorCode == 0 and result12.ErrorCode == 0 and
636             result13.ErrorCode == 0):
637
638             # Вычисляем компоненты
639             c_ya_f = result.Value
640             c_ya_1 = result2.Value * K_aa
641             c_ya_2 = result6.Value * K_aa_rl
642
643             # Суммарный коэффициент для разных углов атаки
644             row = [M]
645             for angle_deg in angles_deg:
646                 angle_rad = angle_deg * math.pi / 180.0
647                 c_ya_1_sum = (c_ya_f * S_f_bar +
648                             c_ya_1 * S_1_bar * result12.Value +
649                             c_ya_2 * S_2_bar * result13.Value) * angle_rad
650                 row.append(c_ya_1_sum)
651
652             writer14.writerow(row)
653             M += 0.1
654
655 if __name__ == "__main__":
656     main()
657 print(f'Расчет успешно завершен')
658

```

Листинг 1: Полный код программы расчета (main.py)

## Описание основных модулей

### Класс ProbabilityIntegral

Реализует таблицу значений интеграла вероятностей с линейной интерполяцией для промежуточных значений.

### Глобальные параметры

- Геометрические характеристики летательного аппарата
- Параметры крыла и рулевых поверхностей
- Коэффициенты интерференции
- Диапазоны чисел Маха и углов атаки для расчетов

### Функция main()

Выполняет последовательный расчет:

1. Носовой части летательного аппарата
2. Изолированного крыла
3. Коэффициентов интерференции

4. Скоса потока
5. Суммарного коэффициента подъемной силы