AE721 - Boundary Layer Theory
Assignment - 03

Ramkumar S. *
SC22M007, M.Tech. Aerospace - Aerodynamics and Flight Mechanics

This is the report generated for the questions that were solved for the assignment 3.

I. Question - 1

Derive the Falkner-Skan equation from the boundary layer equations by assuming $u_e = Ax^m$ where x is streamwise direction.

SOLUTION

The boundary layer governing equations derived from the Navier-Stokes equations that were used in this derivation are given in Equations (1) and (2).

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\frac{\partial^2 u}{\partial y^2} \tag{2}$$

The flow velocity is assumed to be governed by a power-law profile which is given in Equation (3).

$$u_e = Ax^m \tag{3}$$

where A and m are constants. For ease of derivation, the streamfunction Ψ is taken instead of individual velocity components, and its definition is given in Equation (4), along with a transformation variable η so that the number of independent variables reduces to 1, it is given in Equation (5).

$$\Psi = \int u_e F(\eta) dy \tag{4}$$

$$\eta = y\sqrt{\frac{u_e}{\nu x}} \tag{5}$$

Substituting the η definition and integrating results in the equation for streamfunction as given in Equation (6).

$$\Psi = u_e(x)\zeta(x)f(\eta) \tag{6}$$

where,

$$\zeta(x) = \sqrt{\frac{\nu x}{u_e}} \tag{7}$$

Then the definition of velocity components u and v were written in terms of stream function along with transformation as given in Equations (8) and (9).

$$u = \frac{\partial \Psi}{\partial y} = \frac{\partial \Psi}{\partial \eta}\frac{\partial \eta}{\partial y} + \frac{\partial \eta}{\partial x}\cancelto{0}{\frac{\partial x}{\partial y}} = u_e f'(\eta) \tag{8}$$

$$v = -\frac{\partial \Psi}{\partial x} = -\left(\frac{\partial \Psi}{\partial \eta}\frac{\partial \eta}{\partial x} + \frac{\partial \Psi}{\partial x}\cancel{\frac{\partial x}{\partial x}}\right) \tag{9}$$

*SC22M007, M.Tech., Aerospace AFM

Computing further derivatives of Ψ that are required in the momentum equation as given in Equations (10) to (13).

$$\begin{aligned}\frac{\partial^2\Psi}{\partial x\partial y} &= \frac{\partial}{\partial x}\left(\frac{\partial\Psi}{\partial y}\right) \\ &= \frac{du_e}{dx}f'(\eta) - \eta\frac{u_e}{\zeta}\frac{d\zeta}{dx}f''(\eta)\end{aligned} \tag{10}$$

$$\begin{aligned}\frac{\partial\Psi}{\partial x} &= \frac{\partial}{\partial x}\left(u_e\zeta f(\eta)\right) \\ &= \zeta f(\eta)\frac{du_e}{dx} + u_e\frac{d\zeta}{dx}\left(f(\eta) - \eta f'(\eta)\right)\end{aligned} \tag{11}$$

$$\frac{\partial^2\Psi}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial\Psi}{\partial y}\right) = u_e\frac{f''(\eta)}{\zeta} \tag{12}$$

$$\frac{\partial^3\Psi}{\partial y^3} = \frac{\partial}{\partial y}\left(\frac{\partial^2\Psi}{\partial y^2}\right) = \frac{u_e}{\zeta}f'''(\eta) \tag{13}$$

And from Bernoulli equation, the expression for $\frac{dp}{dx}$ is obtained as Equation (14).

$$\begin{aligned}p + \frac{1}{2}\rho u_e^2 &= Constant \\ \frac{dp}{dx} &= -\rho u_e\frac{du_e}{dx}\end{aligned} \tag{14}$$

Now, substituting the Equations (8) to (14) into Equation (2) gives Equation (15).

$$\begin{aligned}\frac{\partial\Psi}{\partial y}\frac{\partial^2\Psi}{\partial x\partial y} - \frac{\partial\Psi}{\partial x}\frac{\partial^2\Psi}{\partial y^2} &= u_e\frac{du_e}{dx} + \nu\frac{\partial^3\Psi}{\partial y^3} \\ \nu\frac{u_e}{\zeta^2}f'''(\eta) + \left[u_e\frac{du_e}{dx} + \frac{u_e^2}{\zeta}\frac{d\zeta}{dx}\right]f(\eta)f'(\eta) + \left(1 - (f'(\eta))^2\right)u_e\frac{du_e}{dx} &= 0.\end{aligned} \tag{15}$$

Further simplification of Equation (15) by substituting Equation (7) gives the final Falkner-Skan equation as Equation (16).

$$\boxed{f'''(\eta) + \left(\frac{m+1}{2}\right)f(\eta)f''(\eta) + \left(1 - (f'(\eta))^2\right)m = 0} \tag{16}$$

II. Question - 2

Solve the Falkner-Skan equation numberically with appropriate boundary conditions for m = 2.0,1.0,0.6,0.3,0.0,-0.05,-0.08,-0.09043.

SOLUTION

The Falkner-Skan equation is again given in Equation (17).

$$f''' + \left(\frac{m+1}{2}\right) ff'' + \left(1 - (f')^2\right) = 0 \tag{17}$$

The appropriate boundary conditions are given below.

location	conditions
$\eta = 0$	$f = 0$ & $f' = 0$
$\eta = \infty$	$f' = 1$ & $f'' = 0$

It is a 3rd order ordinary difference equation. Hence to solve it numerically, the following two approaches have been followed.

1) Finite-difference method
2) Shooting method

A. Finite Difference Method

In this method, following [1], the Equation (17) has split into one 1st order and one 2nd order equations, as given in Equations (18) and (19), this is done due to the limitation in the boundary conditions available.

$$f' = z \tag{18}$$

$$z'' + \frac{m+1}{2} fz' + m\left(1 - (z)^2\right) = 0 \tag{19}$$

Here, the $\eta_{max} = 10$ i.e. finite domain is considered and the boundary conditions for these equations will be

location	conditions
$\eta = 0$	$f = 0$ & $z = 0$
$\eta = \infty$	$z = 1$

Further, Equations (18) and (19) are written in their difference forms using 2nd order central difference scheme as given in Equations (20) and (21).

$$\frac{f_{i+1} - f_i}{2\frac{\Delta\eta}{2}} = \frac{1}{2}\left(z_i + z_{i+1}\right) \tag{20}$$

$$\frac{z_{i-1} - 2z_i + z_{i+1}}{\Delta\eta^2} + \left(\frac{m+1}{2}\right) f_i \frac{z_{i+1} - z_{i-1}}{2\Delta\eta} + m\left(1 - z_i^2\right) = 0 \tag{21}$$

Then the Equations (20) and (21) are rearranged to the forms which can be programmed as given in Equations (22) and (23).

$$f_{i+1} = f_i + \frac{1}{2}\left(z_i + z_{i+1}\right)\Delta\eta; i = 1, 2, ..., N-1 \tag{22}$$

$$a_i z_i = b_i z_{i+1} + c_i z_{i-1} + d_i; i = 2, 3, 4, ..., N-1 \tag{23}$$

where

$$
\begin{aligned}
a_i &= \left(\frac{2}{\Delta\eta^2} + m z_i\right) \\
b_i &= \left(\frac{1}{\Delta\eta^2} + \frac{m+1}{4\Delta\eta} f_i\right) \\
c_i &= \left(\frac{1}{\Delta\eta^2} - \frac{m+1}{4\Delta\eta} f_i\right) \\
d_i &= m
\end{aligned}
$$

For each iteration, the Equation (22) is marched forward from $\eta = 0$ till $\eta = \eta_{max} = 10$ and the Equation (23) is solved iteratively till convergence.

The solution steps followed are given below.
1) initialize z_i and f_i
2) start outer iteration.
3) solve Equation (22) by marching forward in η direction.
4) solve Equation (23) iteratively using the values of f_i computed in previous step, till convergence.
5) check for convergence using z_i values from previous and present outer iteration.
6) continue outer iteration till convergence.

The Equation (17) is solved using this method for all m values except $m = -0.09043$, this method does not converge for this m value. Hence the next, shooting method is used for $m = -0.09043$.
The *Python* code developed for this analysis is given in Section A

B. Shooting method

In this method, the Equation (17) is split into 3 1st order ODEs as given in Equations (24) to (26).

$$f' = g \tag{24}$$
$$g' = h \tag{25}$$
$$h' = -(1-g^2)m - \left(\frac{m+1}{2}\right) f h \tag{26}$$

Here also, the $\eta_{max} = 10$ i.e. finite domain is considered and the boundary conditions for these equations will be

location	conditions
$\eta = 0$	$f = 0$ & $g = 0$
$\eta = \eta_{max}$	$g = 1$

Here, it can be seen that the initial condition i.e. the value of h at $\eta = 0$ is unknown, but the value of g at $\eta = \eta_{max}$ is known. Hence by shooting method, the initial value of h at $\eta = 0$ is assumed and the solution is proceeded till η_{max}.
Now, the value of g at η_{max} may not match with the boundary conditions, hence their error difference is taken as the key for adjusting the initial h value using *bisection* method. The solution of Equation (17) with $m = -0.09043$ is computed using this method.
The *Python* code developed for this analysis is given in Section B

C. Tables of final iteration values

Here, the first deliverable as the tables of computed values of parameters for each value of m are given in Tables 1 to 8

Table 1 computed values for m = -0.05

η	f	f'	f''
0.0	0.0	0.0	0.21247
0.5	0.0276	0.11241	0.23689
1.0	0.11434	0.23623	0.25729
1.5	0.26521	0.36819	0.26852
2.0	0.48291	0.50238	0.26545
2.5	0.76662	0.63073	0.24491
3.0	1.1112	0.74456	0.20795
3.5	1.50756	0.83693	0.16049
4.0	1.944	0.90479	0.11145
4.5	2.40845	0.94952	0.06915
5.0	2.89043	0.97583	0.03815
5.5	3.38218	0.98959	0.01868
6.0	3.87879	0.99598	0.0081
6.5	4.37754	0.99861	0.00311
7.0	4.87713	0.99957	0.00106
7.5	5.37701	0.99988	0.00032
8.0	5.87697	0.99997	9e-05
8.5	6.37697	0.99999	2e-05
9.0	6.87696	1.0	0.0
9.5	7.37696	1.0	0.0
10.0	7.87696	1.0	0.0

Table 2 computed values for m = -0.08

η	f	f'	f''
0.0	0.0	0.0	0.09964
0.5	0.01413	0.0598	0.13949
1.0	0.0631	0.13925	0.17783
1.5	0.15644	0.2369	0.21171
2.0	0.30252	0.34949	0.2367
2.5	0.50748	0.47128	0.24766
3.0	0.77399	0.59416	0.24056
3.5	1.10024	0.70872	0.21472
4.0	1.47987	0.80644	0.17429
4.5	1.90294	0.88196	0.12751
5.0	2.35794	0.93442	0.08349
5.5	2.83402	0.967	0.04871
6.0	3.32251	0.98502	0.02525
6.5	3.81751	0.99388	0.01162
7.0	4.31557	0.99776	0.00475
7.5	4.81488	0.99926	0.00172
8.0	5.31467	0.99978	0.00056
8.5	5.81461	0.99994	0.00016
9.0	6.31459	0.99999	4e-05
9.5	6.81459	1.0	1e-05
10.0	7.31459	1.0	0.0

Table 3 computed values for m = 0.3

η	f	f'	f''
0.0	0.0	0.0	0.72548
0.5	0.0844	0.32522	0.57542
1.0	0.3127	0.57579	0.42788
1.5	0.64821	0.75504	0.29212
2.0	1.05724	0.87179	0.17972
2.5	1.51184	0.93989	0.09815
3.0	1.99162	0.975	0.04704
3.5	2.48364	0.99085	0.01962
4.0	2.98087	0.99707	0.00709
4.5	3.48004	0.99918	0.00221
5.0	3.97982	0.9998	0.00059
5.5	4.47976	0.99996	0.00014
6.0	4.97975	0.99999	3e-05
6.5	5.47975	1.0	0.0
7.0	5.97975	1.0	0.0
7.5	6.47975	1.0	0.0
8.0	6.97975	1.0	0.0
8.5	7.47975	1.0	0.0
9.0	7.97975	1.0	0.0
9.5	8.47975	1.0	0.0
10.0	8.97975	1.0	0.0

Table 4 computed values for m = 0.6

η	f	f'	f''
0.0	0.0	0.0	0.97514
0.5	0.10943	0.41353	0.68262
1.0	0.39034	0.68916	0.42917
1.5	0.77986	0.85319	0.23864
2.0	1.23045	0.93904	0.1156
2.5	1.7111	0.97801	0.04815
3.0	2.20452	0.99317	0.01707
3.5	2.70261	0.99819	0.00511
4.0	3.20213	0.99959	0.00128
4.5	3.70203	0.99992	0.00027
5.0	4.20202	0.99999	5e-05
5.5	4.70201	1.0	1e-05
6.0	5.20201	1.0	0.0
6.5	5.70201	1.0	0.0
7.0	6.20201	1.0	0.0
7.5	6.70201	1.0	0.0
8.0	7.20201	1.0	0.0
8.5	7.70201	1.0	0.0
9.0	8.20201	1.0	0.0
9.5	8.70201	1.0	0.0
10.0	9.20201	1.0	0.0

Table 5 computed values for m = 0.3

η	f	f'	f''
0.0	0.0	0.0	0.33138
0.5	0.04141	0.16555	0.33027
1.0	0.16525	0.32916	0.32249
1.5	0.36944	0.48597	0.30227
2.0	0.64889	0.62887	0.26671
2.5	0.99473	0.75042	0.21762
3.0	1.39483	0.84536	0.16173
3.5	1.83541	0.91255	0.10819
4.0	2.30325	0.95521	0.06459
4.5	2.78752	0.97935	0.03423
5.0	3.2806	0.99147	0.01605
5.5	3.77787	0.99685	0.00665
6.0	4.2769	0.99896	0.00243
6.5	4.7766	0.9997	0.00078
7.0	5.27652	0.99992	0.00022
7.5	5.7765	0.99998	6e-05
8.0	6.27649	1.0	1e-05
8.5	6.77649	1.0	0.0
9.0	7.27649	1.0	0.0
9.5	7.77649	1.0	0.0
10.0	8.27649	1.0	0.0

Table 6 computed values for m = 1.0

η	f	f'	f''
0.0	0.0	0.0	1.23239
0.5	0.13348	0.49462	0.75845
1.0	0.45903	0.77782	0.39825
1.5	0.88706	0.91614	0.17718
2.0	1.36167	0.9732	0.06596
2.5	1.85412	0.99285	0.02029
3.0	2.35224	0.99842	0.0051
3.5	2.85186	0.99972	0.00104
4.0	3.35179	0.99996	0.00017
4.5	3.85178	1.0	2e-05
5.0	4.35178	1.0	0.0
5.5	4.85178	1.0	0.0
6.0	5.35178	1.0	0.0
6.5	5.85178	1.0	0.0
7.0	6.35178	1.0	0.0
7.5	6.85178	1.0	0.0
8.0	7.35178	1.0	0.0
8.5	7.85178	1.0	0.0
9.0	8.35178	1.0	0.0
9.5	8.85178	1.0	0.0
10.0	9.35178	1.0	0.0

Table 7 computed values for m = 2.0

η	f	f'	f''
0.0	0.0	0.0	1.71443
0.5	0.17405	0.62189	0.82211
1.0	0.5618	0.88627	0.30046
1.5	1.03103	0.97311	0.08461
2.0	1.52442	0.9951	0.01811
2.5	2.02333	0.99933	0.00289
3.0	2.52319	0.99993	0.00034
3.5	3.02318	0.99999	3e-05
4.0	3.52318	1.0	0.0
4.5	4.02318	1.0	0.0
5.0	4.52318	1.0	0.0
5.5	5.02318	1.0	0.0
6.0	5.52318	1.0	0.0
6.5	6.02318	1.0	0.0
7.0	6.52318	1.0	0.0
7.5	7.02318	1.0	0.0
8.0	7.52318	1.0	0.0
8.5	8.02318	1.0	0.0
9.0	8.52318	1.0	0.0
9.5	9.02318	1.0	0.0
10.0	9.52318	1.0	-0.0

Table 8 computed values for m = -0.09043

η	f	f'	f''
0.0	0.0	0.00433	0.0
0.00242	0.01346	0.04953	0.5
0.01722	0.04949	0.09453	1.0
0.05563	0.10783	0.13852	1.5
0.12861	0.18751	0.17946	2.0
0.24631	0.28619	0.21381	2.5
0.41723	0.39942	0.23677	3.0
0.647	0.52022	0.24338	3.5
0.9372	0.63955	0.23065	4.0
1.28468	0.74781	0.19971	4.5
1.68184	0.83721	0.15654	5.0
2.11806	0.90383	0.11021	5.5
2.58196	0.94835	0.06931	6.0
3.06341	0.97491	0.03881	6.5
3.5548	0.98901	0.01931	7.0
4.05119	0.99568	0.00853	7.5
4.54984	0.99848	0.00335	8.0
5.04938	0.99952	0.00117	8.5
5.54925	0.99987	0.00036	9.0
6.04921	0.99997	0.0001	9.5
6.54921	1.0	2e-05	10.0

D. Comparison plots

The second deliverable, a list of comparison plots, have been given in this section.

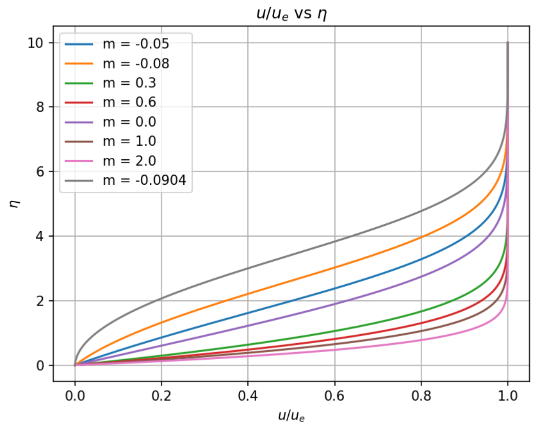


Fig. 1 streamwise velocity profiles plot for different m values

The streamwise velocity profile variations for different m values can be found in Figure 1. From the graph, it can be seen that as the m value increases, the gradient of velocity is also increased. The velocity profile for $m = -0.09043$ shows, the edge of separation.

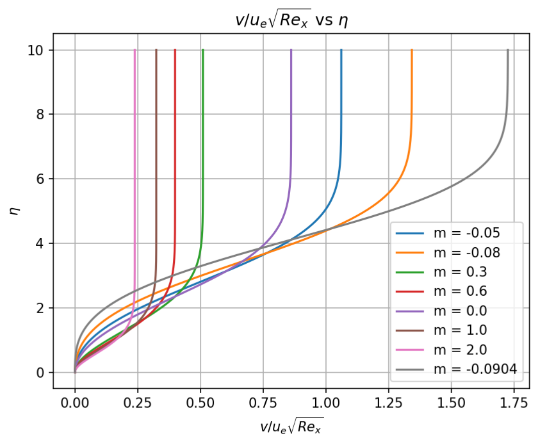


Fig. 2 normal velocity profiles plot for different m values

The normal velocity profiles given in Figure 2 for different m values show that the normal velocity decreases with increase in m value, indicating the change in wedge angle for the given flow.

The shear stress profiles given in Figure 3 show the inflection of profile for the m values less than 0, indicating a favourable shear stress gradient for the flow, till the point of separation where the adverse pressure gradient dominates the shear stress gradient.

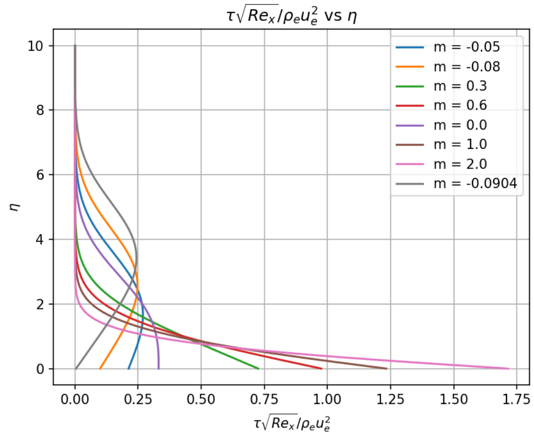


Fig. 3 shear stress profiles for different values of m

E. Table of derived variables

In this section, the computed variables i.e. f, f', f'', f''' were used in obtaining values for derived variables given below.

$\frac{\delta^*}{\delta_{FS}}$:
This parameter gives the nondimensional displacement thickness for the given flow field. The value for this is obtained from solution variables as given in Equation (27).

$$\begin{aligned}\delta^* &= \int \left(1 - \frac{u}{u_e}\right) dy \\ \frac{\delta^*}{\delta_{FS}} &= \int (1 - f')\, d\eta \\ \frac{\delta^*}{\delta_{FS}} &= [\eta - f(\eta)]_0^{\eta_{max}}\end{aligned} \tag{27}$$

Similarly, the nondimensional momentum thickness is obtained by integration and is given by Equation (28) and this has been solved by numerical integration.

$$\frac{\theta}{\delta_{FS}} = \int \left(f' - (f')^2\right) d\eta \tag{28}$$

And the nondimensional shearstress is obtained as Equation (29).

$$\begin{aligned}\tau &= \mu \frac{\partial u}{\partial y} \\ &= \mu u_e f''(\eta)\sqrt{\frac{u_e}{\nu x}} \\ \frac{\tau\sqrt{Re_x}}{\rho_e u_e^2} &= f''(\eta) \\ \frac{1}{2} C_f \sqrt{Re_x} &= f''(\eta)|_{@wall}\end{aligned} \tag{29}$$

The expressions for λ and τ are given in Equations (30) and (31).

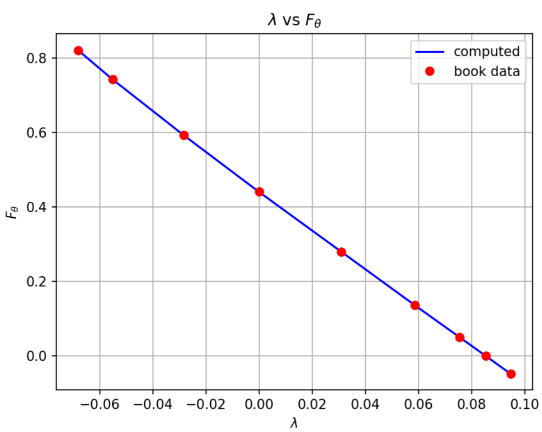
$$\lambda = \frac{\theta^2}{\nu}\frac{du_e}{dx} = m\left(\frac{\theta}{\delta_{FS}}\right)^2 \tag{30}$$

$$\tau = Re_\theta \frac{C_f}{2} = \left(\frac{\theta}{\delta_{FS}}\right)^2 f'' \tag{31}$$

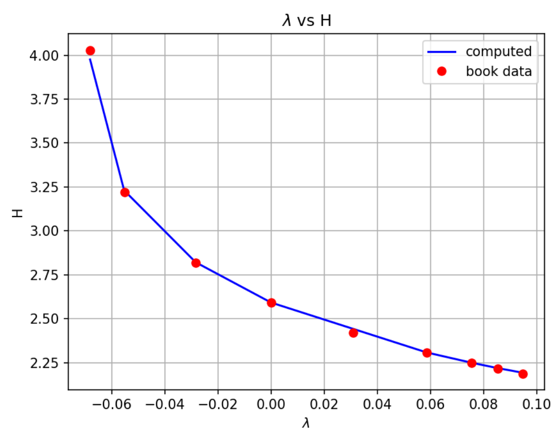
Table 9 Falkner-Skan Equation's derived solution variables

m	$\frac{\delta^*}{\delta_{FS}}$	$\frac{\theta}{\delta_{FS}}$	H	$\sqrt{Re_x}\frac{C_f}{2}$	λ	τ	F_θ
-0.0904	3.45079	0.86796	3.97577	0.00433	-0.0681	0.00376	0.82145
-0.08	2.68541	0.83128	3.23047	0.09964	-0.05528	0.08283	0.74395
-0.05	2.12304	0.75256	2.82108	0.21247	-0.02832	0.15989	0.59283
0.0	1.72351	0.66485	2.59233	0.33138	0.0	0.22032	0.44064
0.3	1.02025	0.44203	2.30809	0.72548	0.05862	0.32069	0.13631
0.6	0.79799	0.35472	2.24962	0.97514	0.0755	0.3459	0.05015
1.0	0.64822	0.29212	2.21902	1.23239	0.08533	0.36	-4e-05
2.0	0.47682	0.21739	2.19343	1.71443	0.09451	0.37269	-0.04728

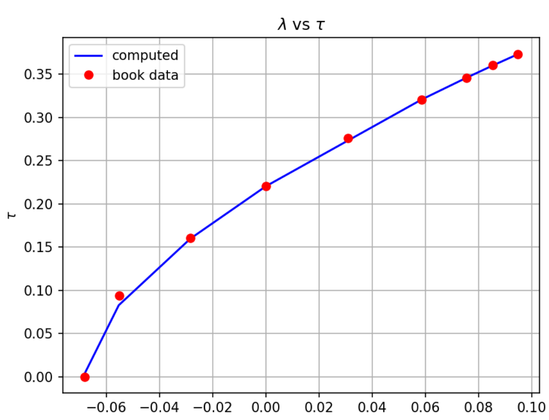
The values obtained for above variabeles for each m value are given in Table 9. And their comparison plots were given in Figure 4, the computed data were compared against the book data given in [2] to show the accuracy of computation.



(a) λ **vs** F_θ



(b) λ **vs H**



(c) λ **vs** τ

Fig. 4 Variation of derived parameters vs λ

III. Question 3

Calculate the value of $\frac{\delta^*}{\delta_{FS}}$, $\frac{\theta}{\delta_{FS}}$,H,$\sqrt{Re_x}\frac{C_f}{2}$, fro the given values of m using analytical Thwaits method and compare with the numerical solution of Falkner Skan equation.

SOLUTION

The procedure followed for computing given variables using Thwait's method is given below.

The given velocity profile is assumed to be governed by power-law, i.e., $u_e(x) = Ax^m$

Then the following equation is used to compute the momentum thickness from the velocity profile.

$$\frac{\theta^2}{\nu} = \frac{0.45}{u_e^6}\int u_e^5 dx$$
$$\frac{\theta^2}{\nu} = \frac{0.45x^{1-m}}{A\,(5m+1)}$$

The nondimensional distance parameter λ is then calculated as follows.

$$\lambda = \frac{\theta^2}{\nu}\frac{du_e}{dx} = \frac{0.45m}{5m+1}$$

Then, the values of H and T are computed using the given approximate expression as

$$H(\lambda) = 2.62 - 4.1\lambda + 14\lambda^3 + \frac{0.56\lambda^2}{(\lambda+0.18)^2}$$
$$T(\lambda) = 0.22 + 1.52\lambda - 5\lambda^3 - \frac{0.072\lambda^2}{(\lambda+0.18)^2}$$

Using, the values of H and T computed above, the following values were computed.

$$H = \frac{\delta^*}{\theta} \rightarrow \delta^* = H\theta$$
$$\frac{\delta*}{\delta_{FS}} = H\sqrt{\frac{0.45}{5m+1}}$$
$$\frac{\theta}{\delta_{FS}} = \sqrt{\frac{0.45}{5m+1}}$$

Lastly, the nondimensional shear stress is computed as follows.

$$\tau = Re_\theta\frac{C_f}{2} \rightarrow \sqrt{Re_x}\frac{C_f}{2} = \frac{T}{\left(\frac{\theta}{\delta_{FS}}\right)}$$

The computed analytical values were compared with the above computed numerical values in Figures 5 to 9.

It can be seen that the Thwait solution does not match well with the numerical solution, but it is sufficient enough for the preliminary computations. The *Python* code developed for this post-processing is given in Section C.

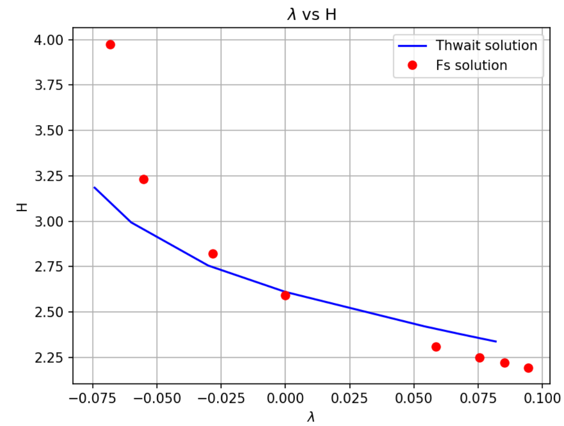


Fig. 5 λ **vs** H

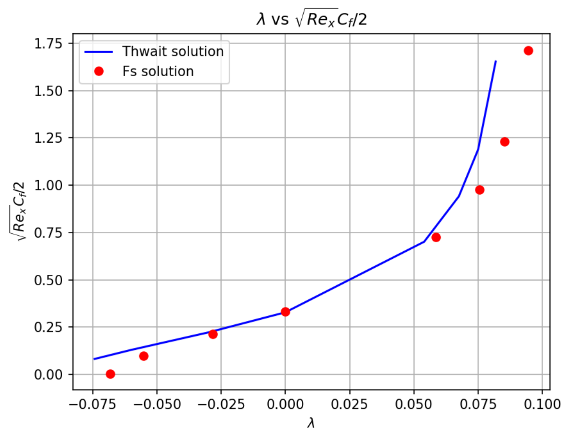


Fig. 6 λ **vs** $\sqrt{Re_x}\frac{C_f}{2}$

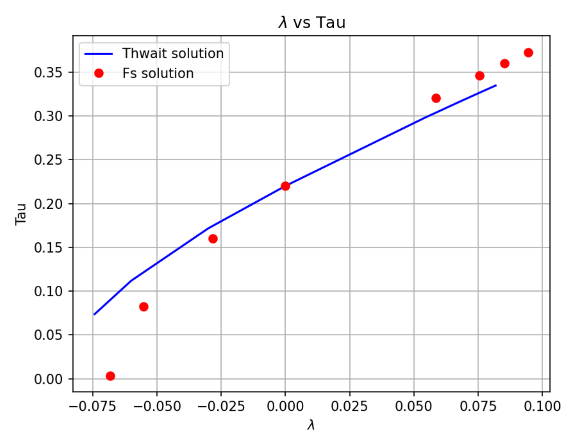


Fig. 7 λ **vs** T

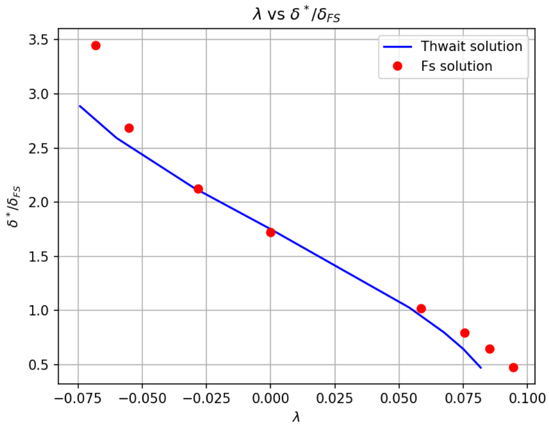


Fig. 8 λ **vs** $\frac{\delta^*}{\delta_{FS}}$

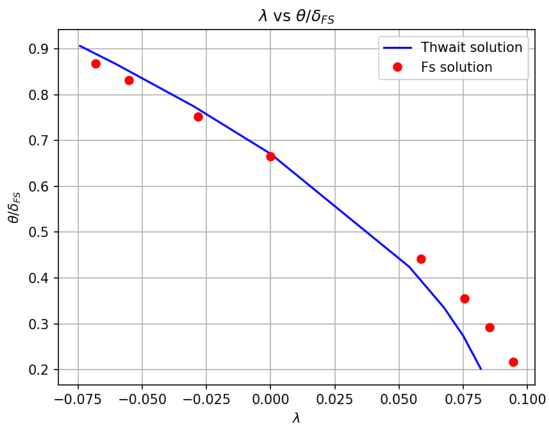


Fig. 9 λ **vs** $\frac{\theta}{\delta_{FS}}$

References

[1] Han, S. "Finite Difference Solution of the Falkner—Skan Wedge Flow Equation." International Journal of Mechanical Engineering Education 41.1 (2013): 1-7.

[2] Drela, Mark. Flight vehicle aerodynamics. MIT press, 2014.

A. Appendix - Finite Difference Method code for FS equation

This section contains the Python code that solves the FS equation using Finite Difference Method.

```python
#!/bin/python3
"""------------------------------------------------------------------------------
Numerical solution of Falkner-Skan wedge flow equation using
finite difference method
------------------------------------------------------------------------------"""

# importing needed modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from copy import copy as cp
import os

# defining computation parameters----------------------------------------------
m_values = [2,1,0.6,0.3,0,-0.05,-0.08]
eta_max = 10.0
N_eta = 201

N_iteration = 101
tolerance = 1e-6

dn = eta_max/(N_eta-1)

# computation variables definition--------------------------------------------
f_list = []; z_list = []

f = np.zeros(N_eta)
#  z = np.zeros(N_eta)

# applying initial and boundary conditions
f[0] = 0
#  z[0] = 0
#  z[N_eta-1] = 1.0
z = np.linspace(0,1.0,N_eta)
eta = np.linspace(0,eta_max,N_eta)

# solution begin----------------------------------------------------------------
for m in m_values:
    # reinitializing conditions
    f = np.zeros(N_eta)
    z = np.linspace(0,1.0,N_eta)
    for itr in range(N_iteration):
        # marching the f equation from lower boundary
        for i in range(N_eta-1):
            f[i+1] = f[i] + 0.5*(z[i]+z[i+1])*dn

        # solving iteratively the z equation
        z_prev = cp(z)

        for j in range(1000):
            for i in range(1,N_eta-1):
                # computing coefficients
                ai = 2/dn**2 + m*z[i]
                bi = 1/dn**2 + (m+1)/4/dn*f[i]
                ci = 1/dn**2 - (m+1)/4/dn*f[i]
                di = cp(m)

                # solving equation
                z[i] = 1/ai*(bi*z[i+1] + ci*z[i-1]+di)
            # convergence check
            convergence = np.max(np.abs(z_prev - z))
            z_prev = cp(z)
            if convergence < tolerance:
                break
```

```
        # status update
        print("m = ",m,"; iteration : ", itr,"; z iteration : ",j,
                "; z convergence = ",convergence)
    f_list.append(f)
    z_list.append(z)

# post processing section------------------------------------------------------
# obtaining computation variables
f_d_list = z_list
f_dd_list = []

# computing f_double dash
f_dd = np.zeros(N_eta)
for i in range(len(f_d_list)):
    z = f_d_list[i]
    f_dd = np.zeros(N_eta)
    for j in range(1,N_eta-1):
        f_dd[j] = (z[j+1] - z[j-1])/dn/2.0
    # linear interpolation on boundaries
    f_dd[0] = 2*f_dd[1] - f_dd[2]
    f_dd[N_eta-1] = 2*f_dd[N_eta-2] - f_dd[N_eta-3]

    print("name=",i,"\n",f_dd)

    # appending to list
    f_dd_list.append(f_dd)

# preparing dataframe to store computed values to csv
# refreshing storage directory
os.system("rm -rf tables_csv && mkdir tables_csv")
# looping through each m_values
for i in range(len(m_values)):
    # preparing data frame
    fid = pd.DataFrame(np.transpose([eta,f_list[i],f_d_list[i],f_dd_list[i]]),
            columns=["eta","f","g","h"])
    # preparing filename
    fname = "tables_csv/data_table_m="+str(m_values[i])+".csv"
    # writing to csv
    fid.to_csv(fname, index = None)


plt.figure()
for i in range(len(m_values)):
    plt.plot(z_list[i],eta,label='m = '+str(m_values[i]))
plt.grid()
plt.legend()
plt.xlabel("$u/u_e$")
plt.ylabel("$\eta$")
plt.title(r"$u/u_e$ vs $\eta$")
plt.savefig("plot_1.png", dpi = 150)

plt.figure()
for i in range(len(m_values)):
    plt.plot(eta/2*z_list[i],eta,label='m = '+str(m_values[i]))
plt.grid()
plt.legend()
plt.xlabel(r"$\left(v \sqrt{Re_x}\right)/u_e$")
plt.ylabel(r"$\eta$")
plt.title(r"$\left(v \sqrt{Re_x}\right)/u_e$ vs $\eta$")
plt.savefig("plot_2.png", dpi = 150)

plt.show()
```

B. Appendix - Shooting Method code for FS equation

This section contains the Python code that solves the FS equation using Shooting Method.

```python
#!/bin/python3
"""------------------------------------------------------------------------
Numerical solution of Falkner-Skan wedge flow equation using
shooting method
------------------------------------------------------------------------"""

# importing needed modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from copy import copy as cp

# defining computation parameters-----------------------------------------
m_values = [-0.0904]
eta_max = 10.0

tolerance = 1e-6

dn = 0.0001

# computation variables definition----------------------------------------
f = []
#   z = np.zeros(N_eta)


# function definitions section--------------------------------------------
# test function definition
def test_func(h_init,m):
    # defining initial conditions
    g0 = 0
    h0 = cp(h_init)
    f0 = 0

    eta = 0
    # computing solution
    while eta <= eta_max:
        # computing derivatives
        dfdn = g0*1 # *1 to prevent absolute referencing
        dgdn = h0*1
        dhdn = -(1 - g0**2)*m - (m+1)/2*f0*h0

        # computing next step values
        f1 = f0 + dfdn*dn
        g1 = g0 + dgdn*dn
        h1 = h0 + dhdn*dn

        # updating values
        f0 = cp(f1)
        g0 = cp(g1)
        h0 = cp(h1)
        eta += dn

    return g1

# solution making function definition
def make_solution(h_init,m):
    # defining lists to store values with initial conditions
    f = [0]
    g = [0]
    h = [h_init]
    eta = [0]

    g0 = g[-1]
    h0 = h[-1]
    f0 = g[-1]

    # computing solution
```

```
    while eta[-1] <= eta_max:

        # computing derivatives
        dfdn = g0*1 # *1 to prevent absolute referencing
        dgdn = h0*1
        dhdn = -(1 - g0**2)*m - (m+1)/2*f0*h0

        # computing next step values
        f1 = f0 + dfdn*dn
        g1 = g0 + dgdn*dn
        h1 = h0 + dhdn*dn

        # appending to the list
        f.append(f1)
        g.append(g1)
        h.append(h1)
        eta.append(eta[-1]+dn)

        # updating values
        f0 = cp(f1)
        g0 = cp(g1)
        h0 = cp(h1)

    print("solution done")

    return f,g,h,eta

# computation section--------------------------------------------------------
# making lists to store computed values
f_list = []
g_list = []
h_list = []

# looping through m values
for i in range(len(m_values)):

    print("solving for m = ",m_values[i])

    # running bisection to compute exact value
    h_a = 0
    h_b = 1
    h_c = (h_a + h_b)/2.0

    while abs(test_func(h_c,m_values[i]) - 1.0) > 1e-6:
        res = test_func(h_c,m_values[i]) - 1.0

        if res < 0:
            h_a = cp(h_c)
        else:
            h_b = cp(h_c)

        h_c = (h_a + h_b)/2.0

    # getting solution for the obtained initial condition
    f,g,h,eta = make_solution(h_c, m_values[i])

    # appending the solution to the list
    f_list.append(f)
    g_list.append(g)
    h_list.append(h)



# post-processing section----------------------------------------------------

# preparing dataframe to save the data
fid = pd.DataFrame(np.transpose([f,g,h,eta]), columns = ["f","g","h","eta"])
fid.to_csv("table_data_m=-0.0904.csv", index = None)
```

```
# plotting graphs
plt.figure()
for i in range(len(m_values)):
    plt.plot(g_list[i], eta, label = "m = "+str(m_values[i]))
plt.grid()
plt.xlabel(r"$u/u_e$")
plt.ylabel(r"$\eta$")
plt.title(r"$u/u_e$ vs $\eta$")
plt.legend()
plt.savefig("plot_1.png", dpi = 150)


plt.show()
```

C. Appendix - Post-processing code

This section contains the Python code that is used for postProcessing the computed data.

```
#!/bin/python3
"""------------------------------------------------------------------------
postprocessing the computed data
------------------------------------------------------------------------"""

# importing needed modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os,glob

# reading files--------------------------------------------------------
fnames = sorted(glob.glob1(os.getcwd()+"/../01_FDM/tables_csv/","*.csv"))
# adding path to all the fnames
for i in range(len(fnames)):
    fnames[i] = "../01_FDM/tables_csv/"+fnames[i]
fnames.append("../02_shootingMethod/table_data_m=-0.0904.csv")

fid = []
for name in fnames:
    fid.append(pd.read_csv(name))

# obtaining m values from the filenames
m = []
for name in fnames:
    m.append( float(name.split("=")[1].split("c")[0][:-1]))

# preparing table data-------------------------------------------------
d_star_dFS_list = [] #  delta*/delta_FS
theta_dFS_list = [] #  theta/delta_FS
H_list = [] #  H
Rex_Cf_list = [] #  sqrt{Re_x} Cf/2
lambda_list = [] #  lambda
Tau_list = [] #  captial Tau
F_theta_list = [] #  F_theta

# looping through m values
for i in range(len(m)):
    # computing d_star_dFS
    d_star_dFS = ((fid[i]['eta'].iloc[-1] - fid[i]['f'].iloc[-1]) -
            (fid[i]['eta'].iloc[0] - fid[i]['f'].iloc[0]))

    # computing theta_dFS numerically
    sum = 0
    func = lambda j: fid[i]['g'].iloc[j] - fid[i]['g'].iloc[j]**2
    for j in range(1,fid[i].shape[0]-1):
        sum += func(j)
    dn = fid[i]['eta'].iloc[1] - fid[i]['eta'].iloc[0]
```

```
    theta_dFS = dn/2.0*(func(0) + func(fid[i].shape[0]-1) + 2*sum)

    # compute H
    H = d_star_dFS/theta_dFS

    # computing Rex_Cf
    Rex_Cf = fid[i]['h'].iloc[0] #  @ wall

    # computing lamda
    lamda = m[i]*theta_dFS**2

    # computing Tau
    Tau = theta_dFS*fid[i]['h'].iloc[0] #  @ wall

    # computing F_theta
    F_theta = 2*(Tau - (H + 2)*lamda)

    # appending them to the list
    d_star_dFS_list.append(d_star_dFS)
    theta_dFS_list.append(theta_dFS)
    H_list.append(H)
    Rex_Cf_list.append(Rex_Cf)
    lambda_list.append(lamda)
    Tau_list.append(Tau)
    F_theta_list.append(F_theta)

# preparing dataframe to save it
df = pd.DataFrame(np.transpose([m,d_star_dFS_list, theta_dFS_list, H_list,
    Rex_Cf_list, lambda_list, Tau_list, F_theta_list]),
    columns = ["m","d_star_dFS","theta_dFS","H","Rex_Cf",
               "lambda","Tau","F_theta"])
df = df.sort_values("m").reset_index(drop=True)
df.to_csv("table_1_FS.csv", index = None)

# plotting graphs--------------------------------------------------------------

# plot 1 : u/ue vs eta
plt.figure()
for i in range(len(fnames)):
    plt.plot(fid[i]["g"],fid[i]["eta"],label="m = "+str(m[i]))
plt.legend()
plt.grid()
plt.xlabel(r"$u/u_e$")
plt.ylabel(r"$\eta$")
plt.title(r"$u/u_e$ vs $\eta$")
plt.savefig("plot_1.png", dpi = 150)

# plot 2 : v sqrt{Rex}/ue vs eta
plt.figure()
for i in range(len(fnames)):
    plt.plot((fid[i]["g"]*fid[i]["eta"] - fid[i]["f"])/2.0,fid[i]["eta"],label="m = "+str(m[i]))
plt.legend()
plt.grid()
plt.xlabel(r"$v/u_e \sqrt{Re_x}$")
plt.ylabel(r"$\eta$")
plt.title(r"$v/u_e \sqrt{Re_x}$ vs $\eta$")
plt.savefig("plot_2.png", dpi = 150)

# plot 3 : tau sqrt{Rex}/(rhoe Ue^2) vs eta
plt.figure()
for i in range(len(fnames)):
    plt.plot(fid[i]["h"],fid[i]["eta"],label="m = "+str(m[i]))
plt.legend()
plt.grid()
plt.xlabel(r"$\tau \sqrt{Re_x}/\rho_e u_e^2$")
plt.ylabel(r"$\eta$")
plt.title(r"$\tau \sqrt{Re_x}/\rho_e u_e^2$ vs $\eta$")
plt.savefig("plot_3.png", dpi = 150)
```

```
# reading book data
df_book = pd.read_csv("book_data.csv")

# plot 4 : lambda vs H
plt.figure()
plt.plot(df['lambda'], df['H'], '-b', label = "computed")
plt.plot(df_book['lambda'], df_book['H'], 'or', label = "book data")
plt.grid()
plt.legend()
plt.xlabel(r"$\lambda$")
plt.ylabel("H")
plt.title(r"$\lambda$ vs H")
plt.savefig("lambda_vs_H.png", dpi = 150)

# plot 5 : lambda vs tau
plt.figure()
plt.plot(df['lambda'], df['Tau'], '-b', label = "computed")
plt.plot(df_book['lambda'], df_book['Tau'], 'or', label = "book data")
plt.grid()
plt.legend()
plt.xlabel(r"$\lambda$")
plt.ylabel(r"$\tau$")
plt.title(r"$\lambda$ vs $\tau$")
plt.savefig("lambda_vs_Tau.png", dpi = 150)

# plot 6 : lambda vs F_theta
plt.figure()
plt.plot(df['lambda'], df['F_theta'], '-b', label = "computed")
plt.plot(df_book['lambda'], df_book['F_theta'], 'or', label = "book data")
plt.grid()
plt.legend()
plt.xlabel(r"$\lambda$")
plt.ylabel(r"$F_\theta$")
plt.title(r"$\lambda$ vs $F_\theta$")
plt.savefig("lambda_vs_F_theta.png", dpi = 150)

# computing analytical solution using thwait's method
lambda_T_list = []
d_star_dFS_T_list = []
H_T_list = []
Tau_T_list = []
theta_dFS_T_list = []
Rex_Cf_T_list = []

for i in range(len(m)):
    # computing lambda
    lamda = 0.45*m[i]/(5*m[i]+1)

    # computing H and Tau
    H = 2.61 - 4.1*lamda + 14*lamda**3 + 0.56*lamda**2/(lamda + 0.18)**2
    Tau = 0.220 + 1.52*lamda - 5.0*lamda**3 - 0.072*lamda**2/(lamda + 0.18)**2

    # computing d_star_dFS
    d_star_dFS = H*np.sqrt(0.45/(5*m[i]+1))

    # computing theta_dFS
    theta_dFS = np.sqrt(0.45/(5*m[i]+1))

    # computing Rex_Cf
    Rex_Cf = Tau/theta_dFS

    # appending them to the list
    lambda_T_list.append(lamda)
    H_T_list.append(H)
    Tau_T_list.append(Tau)
    d_star_dFS_T_list.append(d_star_dFS)
    theta_dFS_T_list.append(theta_dFS)
```

```
        Rex_Cf_T_list.append(Rex_Cf)

    # preparing dataframe to store analytical data
    df2 = pd.DataFrame(np.transpose([m, d_star_dFS_T_list, theta_dFS_T_list, H_T_list,
        Rex_Cf_T_list, lambda_T_list, Tau_T_list]),
        columns = ["m","d_star_dFS","theta_dFS","H","Rex_Cf",
                      "lambda","Tau"])
    df2 = df2.sort_values("m").reset_index(drop=True)
    df2.to_csv("table_2_thwait.csv", index = None)


    # plot 7 : lambda vs d_star_dFS thwait comparison
    plt.figure()
    plt.plot(df2['lambda'], df2['d_star_dFS'], '-b', label = "Thwait solution")
    plt.plot(df['lambda'], df['d_star_dFS'], 'or', label = "Fs solution")
    plt.grid()
    plt.legend()
    plt.xlabel(r"$\lambda$")
    plt.ylabel(r"$\delta^*/\delta_{FS}$")
    plt.title(r"$\lambda$ vs $\delta^*/\delta_{FS}$")
    plt.savefig("lambda_vs_d_star_dFS_thwait.png", dpi = 150)

    # plot 8 : lambda vs theta_dFS thwait comparison
    plt.figure()
    plt.plot(df2['lambda'], df2['theta_dFS'], '-b', label = "Thwait solution")
    plt.plot(df['lambda'], df['theta_dFS'], 'or', label = "Fs solution")
    plt.grid()
    plt.legend()
    plt.xlabel(r"$\lambda$")
    plt.ylabel(r"$\theta/\delta_{FS}$")
    plt.title(r"$\lambda$ vs $\theta/\delta_{FS}$")
    plt.savefig("lambda_vs_theta_dFS_thwait.png", dpi = 150)

    # plot 9 : lambda vs H thwait comparison
    plt.figure()
    plt.plot(df2['lambda'], df2['H'], '-b', label = "Thwait solution")
    plt.plot(df['lambda'], df['H'], 'or', label = "Fs solution")
    plt.grid()
    plt.legend()
    plt.xlabel(r"$\lambda$")
    plt.ylabel(r"H")
    plt.title(r"$\lambda$ vs H")
    plt.savefig("lambda_vs_H_thwait.png", dpi = 150)

    # plot 10 : lambda vs Tau thwait comparison
    plt.figure()
    plt.plot(df2['lambda'], df2['Tau'], '-b', label = "Thwait solution")
    plt.plot(df['lambda'], df['Tau'], 'or', label = "Fs solution")
    plt.grid()
    plt.legend()
    plt.xlabel(r"$\lambda$")
    plt.ylabel(r"Tau")
    plt.title(r"$\lambda$ vs Tau")
    plt.savefig("lambda_vs_Tau_thwait.png", dpi = 150)

    # plot 11 : lambda vs Rex_Cf thwait comparison
    plt.figure()
    plt.plot(df2['lambda'], df2['Rex_Cf'], '-b', label = "Thwait solution")
    plt.plot(df['lambda'], df['Rex_Cf'], 'or', label = "Fs solution")
    plt.grid()
    plt.legend()
    plt.xlabel(r"$\lambda$")
    plt.ylabel(r"$\sqrt{Re_x}C_f/2$")
    plt.title(r"$\lambda$ vs $\sqrt{Re_x}C_f/2$")
    plt.savefig("lambda_vs_Rex_Cf_thwait.png", dpi = 150)
```

```
plt.show()
```

****