Numerical methods
Computer project №3

Gleb Zhdanko

February 2025

General Information

The computations and visualizations were performed on a MacBook Pro (13-inch, 2020, Two Thunderbolt 3 Ports) equipped with a 1.4 GHz Quad-Core Intel Core i5 processor.

I used Python 3.12.4. The following Python libraries were utilized in the implementation:

- `numpy`: for numerical computations,
- `scipy`: for solving linear systems and other numerical methods,
- `sympy`: for symbolic manipulation and validation,
- `matplotlib`: for creating high-quality visualizations.

Problem 1

Problem: Consider a fully developed laminar flow in a channel of height H with a constant pressure gradient $\frac{\partial p}{\partial x}$ and variable viscosity $\mu(y)$ (due to a non-uniform temperature distribution across the channel). The boundary conditions are as follows:

a) Bottom wall at rest: $u|_{y=0} = 0$,

b) Top wall moving with constant velocity U: $u|_{y=H} = U$.

The equation describing the velocity distribution in the channel is given by:

$$\frac{d}{dy}\left(\mu(y)\frac{du}{dy}\right) = \frac{\partial p}{\partial x}, 0 \leq y \leq H.$$

Obtain the velocity distribution, $u(y)$, using:

1) shooting method and

2) direct method using 100 intervals

for the following set of parameters:

i) $H = 1, \frac{\partial p}{\partial x} = -1, U = -1, \mu_0 = 1, \alpha = -0.5$;

ii) $H = 1, \frac{\partial p}{\partial x} = -1, U = -1, \mu_0 = 1, \alpha = 0$;

iii) $H = 1, \frac{\partial p}{\partial x} = -1, U = -1, \mu_0 = 1, \alpha = 0.5$.

For both shooting and direct methods, you should write your own program. The use of library solvers for boundary value problems is not allowed for this problem. Show the solutions for all three cases for both shooting and direct methods. Compare the solutions for all three cases and explain the differences. For each of the three cases (i-iii), compare the solutions using both methods and comment on which method gives a more accurate solution for the given problem.

Solution:

$$\frac{d}{dy}\left(\mu(y)\frac{du}{dy}\right) = \frac{\partial p}{\partial x}$$

$$\frac{d\mu(y)}{dy}u' + \mu(y)u'' = \frac{\partial p}{\partial x} \Leftrightarrow \frac{-\mu_0 \cdot \frac{\alpha}{H}}{(1+\alpha\frac{y}{H})^2}u' + \frac{\mu_0}{(1+\alpha\frac{y}{H})}u'' = \frac{\partial p}{\partial x}$$

Direct Method: We use the central difference scheme to approximate derivatives. Based on the lecture notes, the tridiagonal system coefficients are:

$$A_i = \frac{-\frac{\alpha}{H}}{1+\alpha\frac{y_i}{H}}, \quad B_i = 0, \quad f_i = \frac{\partial p}{\partial x}\frac{1+\alpha\frac{y_i}{H}}{\mu_0}.$$

The discretized equations yield:

$$\alpha_i = \left(\frac{1}{h^2} + \frac{A_i}{2h}\right), \quad \beta_i = \left(-\frac{2}{h^2}\right), \quad \gamma_i = \left(\frac{1}{h^2} - \frac{A_i}{2h}\right).$$

We solve the resulting system of linear equations to obtain $u(y)$. To verify the solution, we compare it with an analytical solution obtained using Wolfram Alpha. The plots are shown in Fig.1.

Shooting Method: We rewrite the second-order ODE as a system of first-order ODEs:

Introducing variables $u_1 = u$ and $u_2 = u'$, we obtain:

$$\begin{cases} u_1' = u_2, \\ u_2' = \frac{\frac{\alpha}{H}}{1+\alpha\frac{y}{H}} u_2 + \frac{\partial p}{\partial x}\frac{1+\alpha\frac{y}{H}}{\mu_0}, \\ u(0) = 0, \\ u(H) = U. \end{cases}$$

We use the shooting method by iteratively adjusting $u_2(0)$. Initial guess is chosen to be 1, since it doesn't matter in that case. To find best $u_2(0)$ we use Newtone-Method specifically it's realisation from the Lecture Notes, where we need to solve the following system of equations:

$$\begin{cases} U' = V, \\ V' = F(x, u, v, U, V), \\ U(0) = 0, \\ V(0) = 1, \end{cases}$$

where $U = \frac{\partial u_1}{\partial s}$, $V = \frac{\partial u_2}{\partial s}$, and $F = U\frac{\partial f}{\partial u_1} + V\frac{\partial f}{\partial u_2}$. We have used Newtone-Method since it's faster then bisect, for example, and a good practice for understanding the material.

After each shooting iteration, we update the initial guess for $u_2(0)$. Convergence is checked by ensuring the difference between successive guesses is below a predefined tolerance or the difference between new and previous solution is less than tolerance.

We use `solve_ivp` from `scipy.integrate` with the `RK45` method, which provides good accuracy for this type of problem. We set `rtol` equal to the tolerance and `atol` to `tolerance/10` to ensure the method operates with sufficient precision.

The plots are shown in Fig. 1.

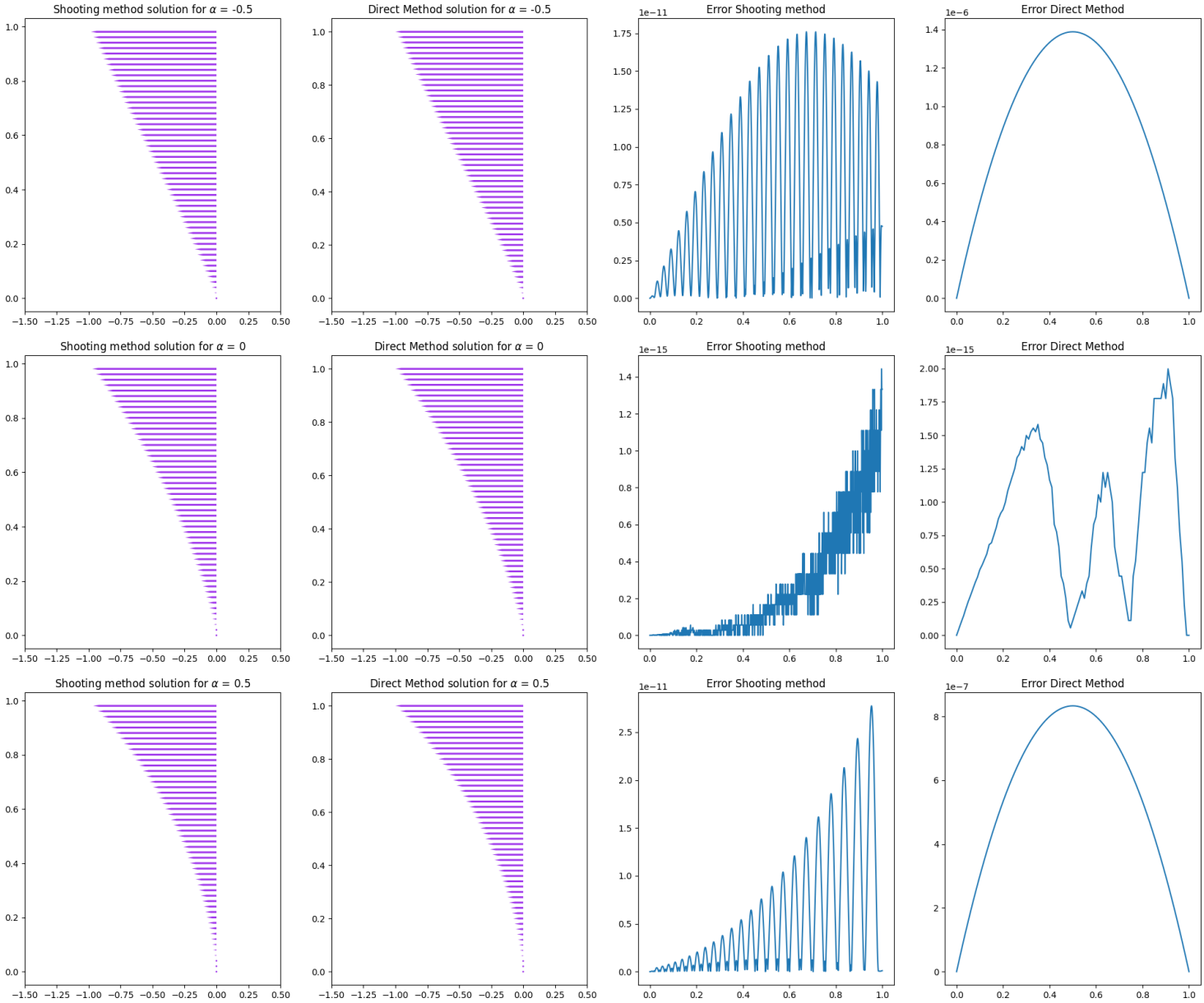
Discussion: From the plots, we observe that for $\alpha = 0$, both methods perform well with high accuracy. Some oscillations in the error plot arise due to numerical precision limits, as the analytical solution is a second-order polynomial and we solve it exact.

For $\alpha = \pm 0.5$, the direct method exhibits a parabolic error distribution, likely due to second-order derivative approximations causing error accumulation towards the center.

The shooting method shows increasing error near the right boundary, which indicates that our method doesn't find exact value for $u_2(0)$.

In general, the shooting method benefits from high tolerance settings but is sensitive to initial conditions. The direct method is more robust and easier to interpret for this problem. By increasing the number of grid points in the direct method we can achieve higher accuracy than shooting method, specifically if solution have some local problems.

Figure 1: Results of first problem



Problem 2

The similarity solution for the incompressible laminar boundary layer is given by

$$f''' + \frac{1}{2} f f'' = 0$$

with the boundary conditions

$$f(0) = f'(0) = 0, \;\; f'(+\infty) = 1,$$

where $(\cdot)' = \frac{d}{d\eta}$, η is the similarity variable, and $f'(\eta)$ corresponds to the tangential velocity component. Solve this problem using the shooting method to obtain the similarity velocity distribution $f'(\eta)$. The use of library solvers for boundary value problems is allowed for this problem.

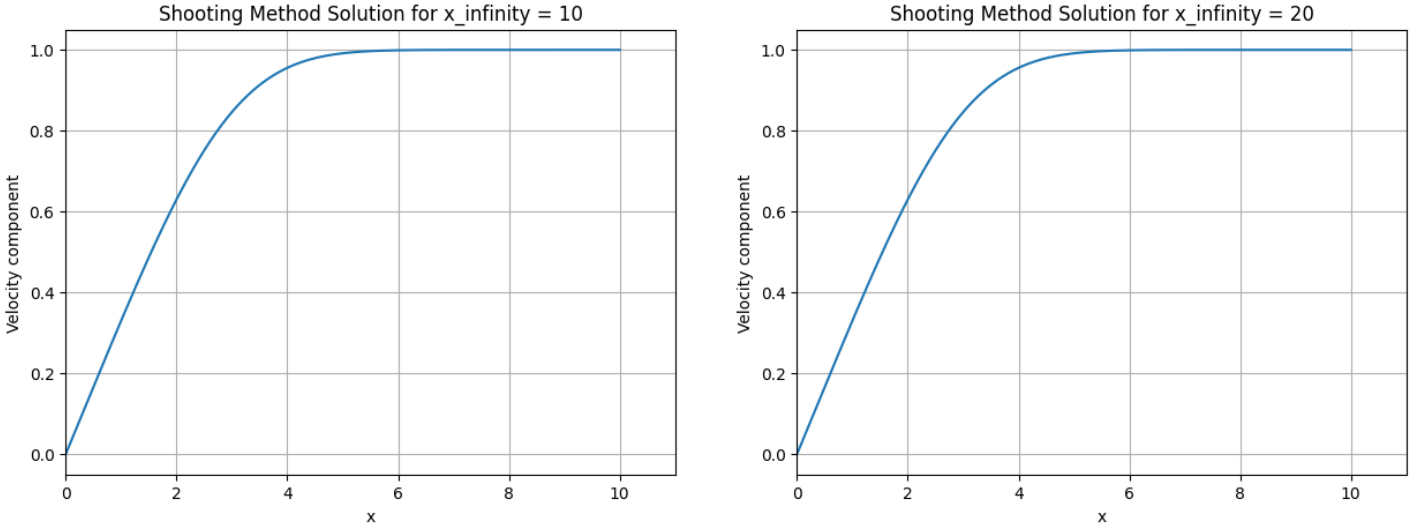
Solution: Let $u_1 = f,\ u_2 = f',\ u_3 = f''$. Then equation transforms to:

$$\begin{cases} u_1' = u_2, \\ u_2' = u_3, \\ u_3' = -\frac{1}{2} u_1 \cdot u_3, \\ u_1(0) = 0, \\ u_2(0) = 0, \\ u_2(+\infty) = 1. \end{cases}$$

I have used realisation of the code from seminar. In general it looks similiar to the first problem (configuraton for `solve_ivp`: method = 'RK45', rtol = 1e-8, atol = 1e-9), however here I have used `root_scalar` with method `bisect` since Newtone Method declined to work properly.

To validate the accuracy of the solution, I compared two solutions with substituting infinity with 10 and 20 for the same guess of $u_3(0) = 0.33205733448266983$ - found by `root_scalar` for infinity equal to 10. The observed difference between these solutions was $4.75720574044658e-16$, confirming that the method performs reliably. The corresponding plots are presented in Fig. 2.

Figure 2: Results of second problem



Problem 3

Consider the problem on the semi-infinite interval $[1, +\infty)$:

$$y''(x) = \frac{4x^2}{(1+x)^2} y(x),\ y(1) = 3,\ y(+\infty) = 0.$$

(a) Approximate a solution to this problem by replacing the boundary condition at $+\infty$ with $y(4) = 0$ and using finite differences for $h = 1/10,\ 1/20$, and $1/40$.

(b) Use a more accurate boundary condition at $x = 4$ and compare the solution with the solution obtained in part (a). Discuss the results.

Solution: I have solved this problem by constructing the finite difference scheme $Ay = b$, where A is a matrix, and y, b are vectors containing all N elements (as discussed in lectures, in contrast to the lecture notes or previous problems). The only difference between (a) and (b) is the last row of matrix A and the corresponding element of b. For case (a), we enforce $A[-1][-1] = 1$ (all other entries in the row are zero), and set $b[-1] = 0$, where index -1 refers to the last row/element.

For case (b), the boundary condition at $x = 4$ is modified by considering the left eigenvectors of the coefficient matrix:

$$A_x = \begin{pmatrix} 0 & 1 \\ \frac{4x^2}{(1+x)^2} & 0 \end{pmatrix} \Rightarrow A_\infty = \begin{pmatrix} 0 & 1 \\ 4 & 0 \end{pmatrix} \Rightarrow$$

$$\Rightarrow \lambda_1 = 2, \lambda_2 = -2$$

$$\Rightarrow (\alpha\ \beta) \begin{pmatrix} -2 & 1 \\ 4 & -2 \end{pmatrix} = (0\ 0) \Rightarrow e_1^T = (2\ 1) \Rightarrow$$

$$\Rightarrow (2\ 1) \begin{pmatrix} y_1(4) \\ y_2(4) \end{pmatrix} = 2y_1(4) + y_2(4) = 0 \Rightarrow$$

$$\begin{cases} y_1' = y_2 \\ y_2' = \frac{4x^2}{(1+x)^2} y_1(x) \\ y_1(3) = 3 \\ 2y_1(4) + y_2(4) = 2y(4) + y'(4) = 0 \end{cases}$$

We can rewrite our problem as:

$$y''(x) = \frac{4x^2}{(1+x)^2} y(x),\ y(1) = 3,\ 2y(4) + y'(4) = 0.$$

To approximate the first derivative, we employ a backward finite difference scheme of second order, to keep the error $O(h^2)$. I wrote a script to automate the derivation of finite difference formulas.[1] Since the focus of this project not on finite difference formulas, the derivation and explanation is not provided. The resulting formula is:

$$f_i' = \frac{3f_i - 4f_{i-1} + f_{i-2}}{2h}$$

Thus, the last row of matrix A is updated as follows:

$$A[-1][-1] = 2 + \frac{3}{2h},\ A[-1][-2] = \frac{-2}{h},\ A[-1][-3] = \frac{1}{2h},$$

with all other elements in the row set to zero, and $b[-1] = 0$.

By solving the linear system, we obtain the numerical solution. The comparison of the two methods is shown in Fig. 3.

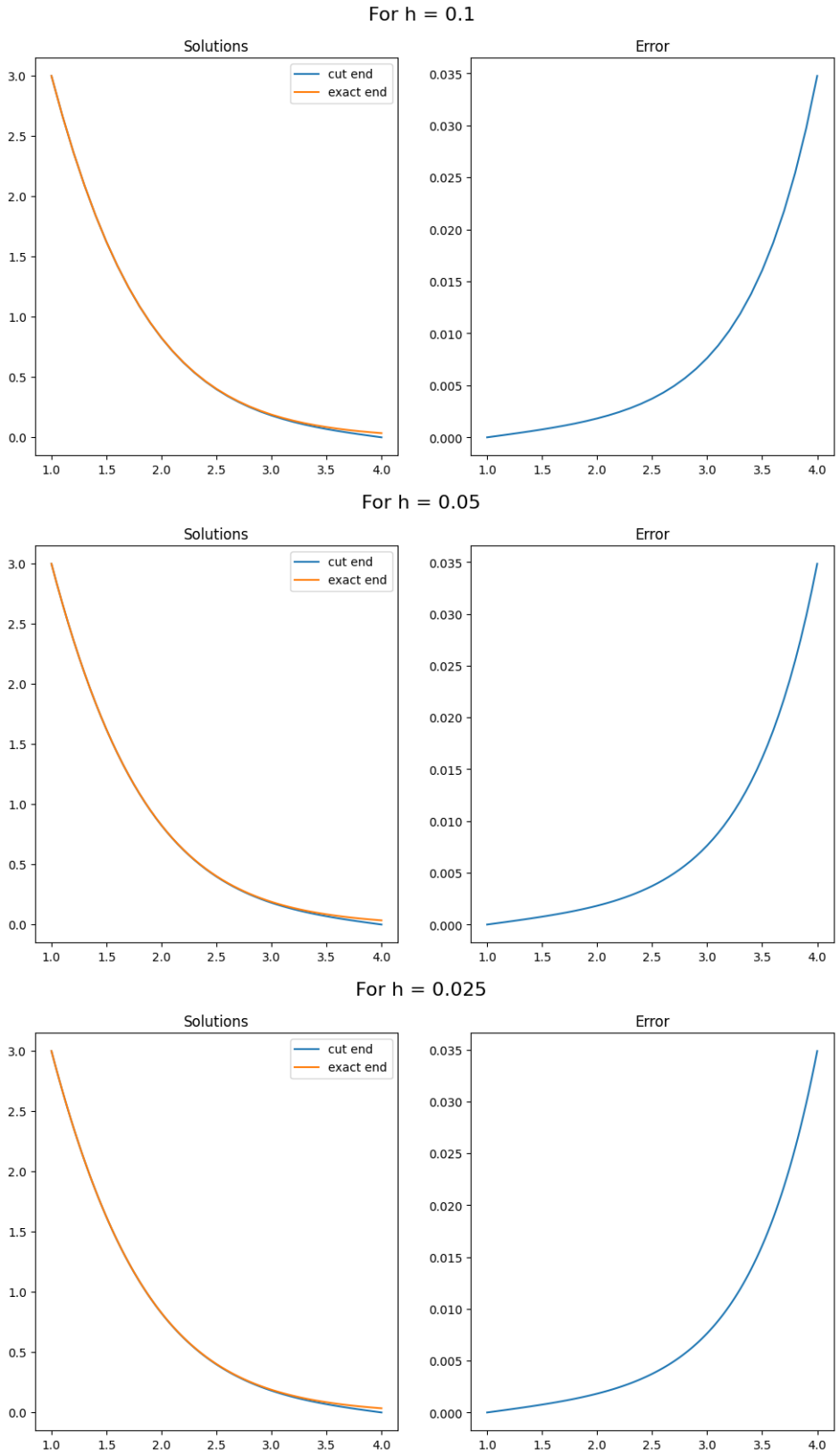
Discussion: It is evident that the error primarily shows near the right boundary. In case (a), replacing the infinite boundary condition with $y(4) = 0$ introduces a loss of accuracy, whereas the method in case (b) is expected to be more precise. Interestingly, the mean and maximum differences between the solutions remain nearly constant for varying h:

- Difference between modified and unmodified BC = 0.00786516848496434 for $h = 0.1$
- Difference between modified and unmodified BC = 0.007710388941699015 for $h = 0.05$
- Difference between modified and unmodified BC = 0.007629932062910776 for $h = 0.025$

[1] `https://github.com/Zhdanko-Gleb/NumericalMethods/blob/main/finite_difference.ipynb`

- Max Difference between modified and unmodified BC = 0.03476692080275041 for $h = 0.1$
- Max Difference between modified and unmodified BC = 0.03484840657288841 for $h = 0.05$
- Max Difference between modified and unmodified BC = 0.034866994144391705 for $h = 0.025$

Figure 3: Results of third problem



Problem 4

The two-dimensional stability of the steady laminar channel flow U(z) is determined by examining the (complex) eigenvalues c of the Orr-Sommerfeld equation:

$$\omega^{iv} - 2\alpha^2\omega'' + \alpha^4\omega = i\alpha Re\left[(U(z) - c)(\omega'' - \alpha^2\omega) - U''(z)\omega\right],$$

with boundary conditions:

$$\omega = \omega' = 0 \; at \; z = \pm 1,$$

where $\omega(z)$ is the vertical perturbation velocity component, α is a perturbation wavenumber, and Re is the Reynolds number. For the plane constant laminar flow $U(z) = 1$ and with parameter values $\alpha = 1$ and $Re = 100$, obtain the first five eigenvalues accurate to 3 significant figures. Use the direct method to discretize the problem and rewrite it as a generalized eigenvalue problem:

$$A\omega = \lambda B\omega,$$

where λ is the eigenvalue and ω is the eigenvector, $\omega = (\omega_0, ..., \omega_N)^T$. Note that numerical discretization introduces spurious eigenvalues. To distinguish spurious eigenvalues from physical ones, solve the problem for two different sets of points, such as $N = 200$ and $N = 201$. Obtain the eigenvalues, sort them in increasing order of magnitude, and compare the two cases $N = 200$ and $N = 201$. If the eigenvalues are close in value, they are considered physical; otherwise, discard them. Spurious eigenvalues are known to jump significantly.

Soltuion: Substituting the specified parameter values into the Orr–Sommerfeld equation leads to:

$$\omega^{(iv)} - 2\omega'' + \omega = 100i\Big[(1 - c)(\omega'' - \omega)\Big].$$

Our objective is to convert this differential equation into a system of linear equations in the form

$$A\,\omega = \lambda\, B\,\omega,$$

where the eigenvalue is related to the complex wave speed c.

We approximate the derivatives using central difference formulas. For the fourth derivative, we use:

$$f_i^{(4)} = \frac{f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}}{h^4},$$

and for the second derivative:

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}.$$

Since the boundary conditions also involve the first derivative, we approximate it with a second-order backward and forward difference formula:

$$f_i' = \frac{3f_i - 4f_{i-1} + f_{i-2}}{2h}, \; f_i' = \frac{4f_{i+1} - 3f_i - f_{i+2}}{2h}.$$

Remark: The derivation of these finite difference formulas was automated using my custom script hosted on GitHub.

After substituting these finite difference approximations into the differential equation, the discrete equation for an interior grid point becomes:

$$\frac{1}{h^4}\omega_{i+2} + \omega_{i+1}\Big(-\frac{4}{h^4} - \frac{2}{h^2} - \frac{100i}{h^2}\Big) + \omega_i\Big(\frac{6}{h^4} + \frac{4}{h^2} + 1 + \frac{200i}{h^2} + 100i\Big)$$
$$+\omega_{i-1}\Big(-\frac{4}{h^4} - \frac{2}{h^2} - \frac{100i}{h^2}\Big) + \frac{1}{h^4}\omega_{i-2} = 100i\; c\left[-\frac{1}{h^2}\omega_{i+1} + \Big(\frac{2}{h^2} + 1\Big)\omega_i - \frac{1}{h^2}\omega_{i-1}\right], i = 2, \ldots N-2.$$

The boundary conditions are:

$$\omega_0 = 0, \quad \omega_N = 0,$$

and using the finite difference approximation for the derivative at the boundaries:

$$\omega_0' = \frac{4\omega_1 - \omega_2}{2h} = 0, \quad \omega_N' = \frac{\omega_{N-2} - 4\omega_{N-1}}{2h} = 0.$$

These yield:

$$\omega_1 = \frac{\omega_2}{4}, \quad \omega_{N-1} = \frac{\omega_{N-2}}{4}.$$

After discretization, the problem is formulated in terms of a 5-diagonal matrix A and a 3-diagonal matrix B. The eigenvalue problem is then solved using the `scipy.linalg.eig` function. To validate the results and to distinguish physical eigenvalues from spurious ones, the algorithm was run for $N = 300$ and $N = 301$. To equalize the number of eigenvalues, zeros were appended to the smaller array at the end (this adjustment proved more accurate than adding to the beginning).

The first five eigenvalues (which showed minimal differences between the two discretizations) were identified as the physical solutions. They are:

$$1. - 0.09313713i, \quad 1. - 0.20569993i, \quad 1. - 0.38943556i, \quad 1. - 0.60045349i, \quad 1. - 0.88274896i.$$

Another Solution: An alternative approach is to write the equation in operator form. Define D^2 as the second derivative operator (approximated by the tridiagonal matrix $[1, -2, 1]/h^2$). Then, the equation can be written as:

$$\Big[(D^2)^2 - 2D^2 + I\Big]\omega = 100i\Big[(1-c)(D^2 - I)\omega\Big].$$

Noting that

$$(D^2 - I)^2\omega = 100i(1-c)(D^2 - I)\omega,$$

we can rearrange this to obtain:

$$\frac{(D^2 - I)^2 - 100i(D^2 - I)}{-100i}\omega = c\,(D^2 - I)\omega.$$

Unfortunately, $(D^2 - I)$ has determinant zero, so we cannot cross it.

Since D^2 is represented by a simple tridiagonal matrix, the remaining steps are analogous to those described above.

For $N = 500$, the six eigenvalues obtained by this method were:

$$\begin{aligned}
&(0.9999999998908486 - 0.03500581177188164j),\\
&(1.0000000000367517 - 0.11001718007110312j),\\
&(1.0000000000367195 - 0.23501590655596402j),\\
&(1.0000000000089708 - 0.40997166337622326j),\\
&(1.000000000002605 - 0.6348419974840832j),\\
&(0.9999999999980635 - 0.9095723374804267j).
\end{aligned}$$

Discussion: It appears that the eigenvalue $(0.9999999998908486 - 0.03500581177188164j)$ is the new value, while the remaining eigenvalues closely match those obtained by the previous formulation. Interesting that previous and this eigenvalues are differ only by imaginary part on around 0.035 which is the the new eigenvalue. That's very interesting, for now I think this is can be due to python error. Because when I make N = 800, the eigenvalues are:

$$\begin{aligned}
&(0.9999999945214997 - 0.034756381286426836j),\\
&(0.9999999838747687 - 0.10902534825126987j),\\
&(1.0000000010490686 - 0.2328056203274987j),\\
&(1.0000000012061436 - 0.4060953074442155j),\\
&(1.0000000035675205 - 0.6288917949828312j),\\
&(0.9999999991612896 - 0.9011916374445035j).
\end{aligned}$$

And for N = 2000:

$$\begin{aligned}
&(0.9999996573615129 - 0.0347102211309181j),\\
&(1.0000002596251434 - 0.10882855111735462j),\\
&(1.0000001433702406 - 0.23236147830848244j),\\
&(0.9999999975263586 - 0.4053100143542976j),\\
&(1.0000000489927583 - 0.6276706135026493j),\\
&(1.0000000470900061 - 0.8994433776421406j).
\end{aligned}$$

It seems that $(0.9999999998908486 - 0.03500581177188164j)$ is spurious eigenvalue, and then our two approaches give almost the same results.

Problem 5

An equation that is obtained in the failure analysis of long suspended structures (such as suspension bridges) by using a nonlinear beam theory is given by

$$y'''' + c^2 y'' + y^+ = 1,$$

where $y^+ = \max\{y, 0\}$. The problem consists of obtaining a nontrivial solution $y(t)$ (the normalized vertical deflection) as a function of $t \in (-\infty, +\infty)$ which is continuous up to the 4-th derivative and that decays exponentially to one as $t \to \pm\infty$. For the parameter c^2 use the value of 1.83297.

Solution: First we need to rewrite conditions of function in a way we can use it. Since our function decays exponentially to one, it means that $|y(t) - 1| \leq A_1 e^{-B_1|x|}, A_1, B_1 - const$, when $t \to \pm\infty$. Using that $y(t) \in C^4(-\infty, \infty)$ we can also conclude that: $y'(\pm\infty) = y''(\pm\infty) = y'''(\pm\infty) = y^{(4)}(\pm\infty) = 0$. Then let's rewrite y^+ using Heaviside step function:

$$y^+ = \theta(y)y, \ \theta(x) = \begin{cases} 1, x > 0, \\ 0, x \leq 0 \end{cases}.$$

Then let's cut our solution to some L, which will be set later. Now we need to solve the following equation:

$$\begin{cases} y'''' + c^2 y'' + \theta(y)y = 1, \\ y(\pm L) = 1, \\ y'(\pm L) = y''(\pm L) = y'''(\pm L) = y^{(4)}(\pm L) = 0. \end{cases}$$

For that I want to use an idea from Shooting method: first guess our solution to define $\theta(y)$ and then solve the equation and modify our guess. The process will be iterative until the change in $|y_{new} - y_{old}| \leq \varepsilon$, y_{new} - solution acquired after substituting y_{old} into $\theta(y)$ and ε - desired tolerance.

We will use finite difference scheme with accuracy $O(h^2)$. The derivatives will be as following:

$$f_i^{(4)} = \frac{f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}}{h^4}$$

$$f_i^{(2)} = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}$$

$$f_i' = \frac{-4f_{i-1} + f_{i-2} + 3f_i}{2h}$$

$$f_i'' = \frac{4f_{i-2} - 5f_{i-1} - f_{i-3} + 2f_0}{h^2}$$

$$f_i''' = \frac{-18f_{i-1} + 24f_{-2} - 14f_{-3} + 3f_{-4} + 5f_0}{2h^3}$$

$$f_i'''' = \frac{-14f_{i-1} + 26f_{i-2} - 24f_{i-3} + 11f_{i-4} - 2f_{i-5} + 3f_i}{h^4}$$

Formulas for forward difference will be the same with change in sign in odd derivatives, however, rhs will be zero, therefore, sign doesn't matter.

After setting everything we can use approach similar to 4th problem and construct matrix A and solve equation

$$Ay_{new} = b.$$

First 5 rows and last 5 rows of matrix A we set manually according to boundary conditions , the rest of the matrix is 5 diagonal. Vector b has the following structure

$$b = [1, 0, 0, 0, 0, 1, \ldots, 1, 0, 0, 0, 0, 1]^T.$$

By setting initial guess, we will derive different solutions(plots are shown on Fig. 4). I have tested $L = 20$ and $L = 40$, I have stopped on $L = 40$. The study of boudary dependence and general convergence of solution isn't provided due to the limit in time. Tolerance was chosen $\varepsilon = 1e - 9$, and number of points in grid $N = 400$. I have tested different initial guesses:

- All 1
- All -1

- alternative sign 1,-1
- middle alternative sign $(1, -1)$ while 1 at the boundaries

Discussion:

The only physical solution which we get was by setting initial guess in the form: in the middle 1,-1 and on the right and left only ones. This is due to the fact that real solution (at least we hope so) have some oscillation in the middle and stable when going to the ends. Other solutions don't have exponential decay to 1 and/or have big amplitudes(bigger than 5) is not physical.

Also, the solution is supposed to be symmetric, that's why we sett such initial guesses.

Figure 4: Results of fifth problem