B1 Engineering Computation: Force on a driven lid over a cavity

Michaelmas Term 2023

Wouter Mostert
wouter.mostert@eng.ox.ac.uk

Last revision: 2023-11-14 11:50:40Z

1 Introduction **2**
1.1 Why do we care about numerical partial differential equations? . . 2
1.2 The physical problem 4
1.3 The mathematical problem 6

2 The numerical problem **9**
2.1 Iterative solution of the linear system of equations 10
2.2 Implementation 12
2.3 Solving the problem 13

3 Your report **16**

1 Introduction

1.1 Why do we care about numerical partial differential equations?

In this project, you will solve a partial differential equation using a numerical method, in order to generate a result which is relevant to an engineering problem. But before we talk about how to go about doing this, let's briefly discuss why we would want to.

By this point in your engineering programme, you have completed two years of courses involving various kinds of mechanics (structural, solid, fluid, and others) as well as learning an array of techniques in mathematical analysis. In virtually all of the work you've done so far, you have estimated loadings, flow rates, temperatures, heat fluxes, distributions, and so on, using mostly analytical techniques. These sorts of approaches usually make various assumptions, which may or may not be reasonable, in order to make an engineering estimate. One might feel sometimes that so many approximations are being made that the reliability of the result is compromised. But one great advantage of such an approach is that it is relatively quick and flexible. For example, in your fluid mechanics problem sheets, you considered many different flow configurations, associated with many different engineering scenarios, but you used a small number of principles and analytical techniques to come up with reasonable answers to problems. These answers may not have been perfectly accurate, but they should have at least given a ballpark figure for the problem of interest.

So, in the context of an engineering design problem, you are being and have been equipped with the tools to be able to iterate quickly on initial designs, to quickly test whether a given concept makes sense and is based in sound engineering principles, both qualitative and quantitative. Let us say that your analysis has shown a particular design to be effective in achieving your a set of goals, and that your team has decided to proceed with that design. Now comes the time to begin to relax your assumptions, and introduce more accuracy into the analysis. You are not yet at the stage where you have a prototype and you can test it under design conditions, but you are past the point where you can do rough estimates.

This is where numerical analysis shines. It is a way of solving a problem in order to, firstly, check whether your initial estimates were reasonable, and secondly,

refine those estimates and provide a best guess for the behaviour of your design under operating conditions. Very often, numerical analysis involves solving a partial differential equation (PDE).

In mechanics, PDEs show up a lot. One of the most commonly encountered PDEs which are tackled numerically is the Poisson equation,

$$\nabla^2\psi = \omega, \tag{1}$$

which can be written explicitly in two-dimensional Cartesian coordinates as,

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = \omega(x, y), \tag{2}$$

where $\psi = \psi(x, y)$. The quantity $\omega(x, y)$ is called a *source term* and is usually known. When $\omega(x, y) = 0$, (2) is called the *Laplace equation*, which you previously encountered in A1. In A1 you were able to solve the Laplace equation because you worked with boundary conditions (BC)s that were relatively straightforward to handle with pen and paper. But if your BCs are not straightforward, or if you have a source term which is difficult to incorporate in the analysis, numerical approaches become relevant.

The following project is based on a presentation of numerical methods for fluid mechanics in the first two chapters of the text *Think before you compute*, by E.J. Hinch (Cambridge University Press, 2020). It is not identical to the problem presented there, and is simpler in many ways, but the discussion in the text may help you get some insight into the significance of various aspects of the numerical approach. In any case though, the problem as presented in this document is self-contained - you should not need reference to any materials outside of this and your other B1 material.

Also, in the following there will be some "boxed" items. These are landmark points for your special attention. They may ask you to do specific tasks - although you may or may not choose to include the outcomes of these tasks in your report. The requirements for your report are discussed at the end of the document.

1.2 The physical problem

The basic question: Consider a two-dimensional square box of side length L_0 filled with a fluid with a density ρ and dynamic viscosity μ, for example an industrial lubricant. This box has a lid that moves at speed U_0, and which is contact with the fluid. If you slide the lid over the box, what is the frictional force F_f on the lid from the fluid?

To answer this question, think about how you were equipped to solve it from P4. In that course you learned that the viscous stress induced by a velocity gradient in the fluid is given by (in two dimensions),

$$\tau(x, y) = \mu \frac{\partial}{\partial y} u(x, y), \tag{3}$$

where μ is the dynamic viscosity of the fluid. Let the location of the bottom left corner of the box be $x = 0, y = 0$. If the lid, located at $y = L_0$, is moving horizontally over the top of the cavity where the fluid is, the no-slip condition implies that the velocity at the lid must be zero relative to the lid i.e. the lid "pulls" the fluid along with it. The velocity at points away from the lid is not necessarily zero, so we expect the velocity gradient at the lid to be nonzero. From (3), this causes a shear stress on the lid, which when integrated along the length of the lid L_0, sums to the frictional force F_f,

$$F_f = \int_0^{L_0} \tau(x, L_0) dx = \mu \int_0^{L_0} \left. \frac{\partial u}{\partial y} \right|_{y=L_0} dx. \tag{4}$$

So you really need to find $\partial u / \partial y$ along the lid.

You can't do this by pen and paper because even though the problem only really cares about the friction at the lid, the answer that you get will depend on what's happening everywhere within the cavity, and there is no obvious way to intuit what that flow will look like. Yes, you might expect streamlines to form closed loops which travel roughly parallel to the lid at the top, but there is not enough information to determine e.g. the equations that would describe the shape of these loops.

Nevertheless, if I were asking you to do this from scratch, the problem as stated above has given you enough information to proceed with a *numerical analysis* in order to answer the question. H owever, doing it from scratch is a bit too much

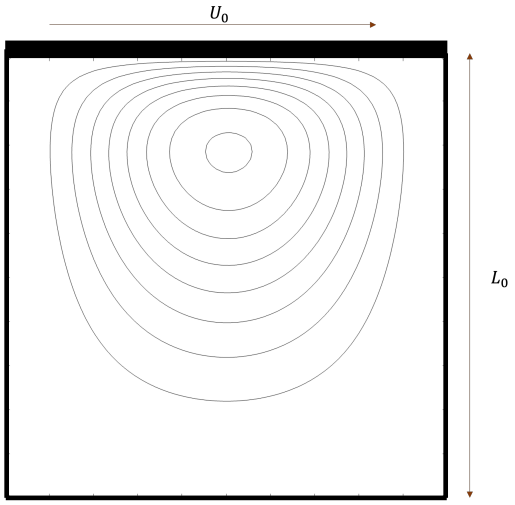

Figure 1. Lid-driven cavity problem

work[1], so instead we provide some extra information that will make it a lot easier.

The question you're answering: Consider a two-dimensional box of side length L_0 filled with a fluid with a density ρ and dynamic viscosity μ, for example an industrial lubricant. This box has a lid that moves at a speed U_0, and which is contact with the fluid. Given data on the vorticity field throughout the cavity, $\omega(x, y)$, what is the frictional force F_f on the lid from the fluid?

In this problem we will set $U_0 = 1m/s$, $L_0 = 1m, \rho = 1kg/m^3$, and set a Reynolds number,

$$\mathrm{Re} \equiv \frac{\rho U_0 L_0}{\mu} = 5. \tag{5}$$

The vorticity is defined by

$$\omega = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}. \tag{6}$$

Tantalisingly, it features $\partial u_x/\partial y$, which is what you need to plug into (4). But you can't use this directly because in general you don't know what $\partial u_y/\partial x$ is. Instead,

[1]Also, to make the problem strictly well-posed mathematically, it is necessary to slightly abstract the interpretation of the moving lid in formulating the boundary conditions for the vorticity advection-diffusion equation (which you don't have to solve here). In the interests of everybody's sanity I will leave it out, but you can read more about it in the reference text if you're interested.

let us define a *streamfunction* $\psi(x, y)$ for which[2],

$$\frac{\partial \psi}{\partial x} = u_y, \quad \frac{\partial \psi}{\partial y} = -u_x. \tag{7}$$

Then it turns out (see below) that,

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \omega(x, y), \tag{8}$$

which is (2). This we can solve numerically, with the appropriate boundary conditions, to determine $\psi(x, y)$, and then we can evaluate (7) to get u_x, numerically compute $\partial u_x / \partial y$, and plug that into (4) to get the desired answer.

For this lid-driven cavity problem, the boundary conditions we need to solve (2) are the following,

$$\psi(x, 0) = C_0, \quad \psi(x, 1) = C_1, \tag{9}$$

$$\psi(0, y) = C_2, \quad \psi(1, y) = C_3, \tag{10}$$

where C_0, C_1, C_2, C_3 are constants. This type of boundary condition, where ψ takes a specified value on the boundary (rather than e.g. specifying its derivative on the boundary) is called a *Dirichlet* condition. This brings us to the first question:

> To solve the Poisson equation (2) for the streamfunction in the lid-driven cavity problem, what values must C_0, C_1, C_2, C_3 take, and what is the physical meaning of these values?

1.3 The mathematical problem

If you are in a hurry, you can safely skip this subsection, but it provides a bit of context that could be useful to you when you come to making sense of your problem.

The big equation of fluid mechanics is the Navier-Stokes equation. For incompressible flow in the absence of gravity, it looks like this,

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{\nabla p}{\rho} + \nu \nabla^2 \mathbf{u}. \tag{11}$$

[2] If you are comparing with your A4 notes, you may notice the signs are different compared to this; this document uses a different sign convention.

Here, $\mathbf{u}$ is the velocity vector, which varies in space and time; ρ is the density (fixed for incompressible flow), $\nu = \mu/\rho$ is the kinematic viscosity, and p is the pressure field, again varying in space and time. This is essentially a statement of conservation of momentum, namely Newton's second law. It is a tough equation to solve, partly because both $\mathbf{u}$ and p are unknown functions. This is significantly complicated by the fact that $\mathbf{u}$ is a vector, and things would be much easier if we could write it in terms of a scalar.

If we assume incompressible, two-dimensional flow, then the conservation of *mass* law looks like this:

$$\nabla \cdot \mathbf{u} = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} = 0. \tag{12}$$

It turns out that the definition for the streamfunction (7) satisfies (12) (try it! Plug one into the other). Crucially, this means we can also simplify (11). If we define the vorticity as,

$$\omega = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}, \tag{13}$$

then ω and ψ are related by the Poisson equation!

$$\nabla^2 \psi = \omega. \tag{14}$$

And it turns out that (11) can be written as,

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \nabla^2 \omega. \tag{15}$$

The pair of equations (14), (15) is called the *streamfunction-vorticity* formulation of fluid mechanics, and they are much easier to handle in a numerical solver than (11) in its raw form. Both of these equations are very "standard" ((15) is called an "advection-diffusion" equation), unlike the full Navier-Stokes equation, and thus can be tackled by standard numerical approaches. A key feature is that if you know the vorticity ω, then you can figure out what ψ is by solving the Poisson equation (14), which you can use to determine $\mathbf{u}$ by plugging into (7). **This is what you will be doing in this project.** You will not be asked to solve (15), but will be given some data for $\omega(x, y)$ so you need only work with (14).

One physical insight: if the fluid is inviscid, then $\nu = 0$. If the fluid is initially

at rest, and the lid on the box starts from rest, then (15) predicts that ω would stay zero, and (14) states (under the BCs we will use) that so would ψ, and that therefore so would $\mathbf{u}$. Since $\mathbf{u}$ stays zero, it cannot develop any shear stresses, and therefore cannot exert a frictional force on the sliding lid. So if the fluid is inviscid, then the lid can freely slide over the fluid and there would be no resistance to its motion.

2 The numerical problem

To solve (2), we will use a finite difference approach. Your lectures will have covered finite difference methods in general, so refer to them for the fundamentals. For this problem, you will use a finite-difference form of the Laplacian operator ∇^2. But first, let's discretise the physical domain.

Let the physical domain be described by $x \times y \in [0, L_0] \times [0, L_0]$. Consider now $(N+2) \times (N+2)$ points spaced evenly over the domain, including the boundaries, lying on the location,

$$x_i = i\Delta x, \quad i = 0, 1, 2, ..., N, N+1, \tag{16}$$

$$y_i = i\Delta y, \quad i = 0, 1, 2, ..., N, N+1, \tag{17}$$

where $\Delta x = \Delta y = L_0/(N+1)$. These points are arranged in a rectangular pattern aligned with the Cartesian axes. Clearly then, there are $N \times N$ points in the interior (i.e. not including the boundaries) of the domain. Our goal is to find the value of $\psi_{i,j}$ at each of these points - i.e. there are $N \times N$ unknowns.

Now a discrete form of (2) is,

$$\frac{\psi_{i-1,j} + \psi_{i,j-1} + \psi_{i+1,j} + \psi_{i,j+1} - 4\psi_{i,j}}{\Delta x^2} = \omega_{i,j}, \quad i, j = 1, 2, ..., N. \tag{18}$$

where the Laplacian operator on the left is using a second-order finite difference formula. There are $N \times N$ such equations (since there are N choices of i, and N choices of j for which we can write (18)). Therefore, by converting from the differential equation (2) to the finite difference equations (18), we have transitioned from a PDE problem into an algebraic problem, a linear system of equations. If we accumulate all $N \times N$ expressions of (18) we obtain,

$$A\Psi = b, \tag{19}$$

where $\Psi = \{\psi_{1,1}, \psi_{1,2}, ..., \psi_{N,N-1}, \psi_{N,N}\}^T$ is the unknown vector of length N^2, and $\mathbf{b}$ is a known vector containing the vorticity data and boundary data (also of length N^2). The matrix A contains the coefficients arising from putting together all of

the expressions of (18). For example, for $N = 3$, we get,

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} \psi_{1,1} \\ \psi_{1,2} \\ \psi_{1,3} \\ \psi_{2,1} \\ \psi_{2,2} \\ \psi_{2,3} \\ \psi_{3,1} \\ \psi_{3,2} \\ \psi_{3,3} \end{pmatrix} = \begin{pmatrix} \Delta x^2 \omega_{1,1} - C_0 - C_2 \\ \Delta x^2 \omega_{1,2} - C_0 \\ \Delta x^2 \omega_{1,3} - C_0 - C_1 \\ \Delta x^2 \omega_{2,1} - C_2 \\ \Delta x^2 \omega_{2,2} \\ \Delta x^2 \omega_{2,3} - C_1 \\ \Delta x^2 \omega_{3,1} - C_2 - C_3 \\ \Delta x^2 \omega_{3,2} - C_3 \\ \Delta x^2 \omega_{3,3} - C_1 - C_3. \end{pmatrix} \tag{20}$$

$N = 3$ is pitifully "low resolution", but it was still pretty tedious to type this out. In this assignment you will need to solve problems for which $N = 20$ or more; in full CFD applications one might need anywhere between $N = 2^7$ to 2^{16}. Needless to say, this is not something you want to solve by hand. But to solve for Ψ, you essentially need to get the inverse of the coefficient matrix A^{-1}, so that $\Psi = A^{-1}\mathbf{b}$. This can be done directly (even in an automated i.e. numerical way), but as N grows to be large this is not the most efficient. Instead, you should adopt an iterative approach.

2.1 Iterative solution of the linear system of equations

The simplest iterative method to solve (19) is the Jacobi method. Let's say you start with an initial guess for the values of Ψ. Perhaps you might guess that $\psi^0_{i,j} = 1$, where the superscript-0 indicates that the initial guess. Then, the Jacobi formula for the $(k+1)$th iterate is an adaptation of (18),

$$\psi^{k+1}_{i,j} = \frac{1}{4}\left(\psi^k_{i+1,j} + \psi^k_{i-1,j} + \psi^k_{i,j+1} + \psi^k_{i,j-1} - \Delta x^2 \omega_{ij}\right), \quad i,j = 1,2,...,N, \tag{21}$$

where $k = 0, 1, 2, ...$ In other words, you sweep across all N^2 values of the grid, and use the old values of ψ at those points to update the new estimate. All the terms on the right hand side are old, and only the term on the left is new. This sweep is repeated on the numerical grid until it converges. In matrix form, this is equivalent to writing $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$, where $\mathbf{D}$ is a matrix containing only

the diagonal terms of $\mathbf{A}$; $\mathbf{L}$ contains only the remaining values of $\mathbf{A}$ in the upper triangle (not including the diagonal), and $\mathbf{L}$ contains only the remaining values in the lower triangle (not including the diagonal). So (19) is

$$(\mathbf{D}+\mathbf{L}+\mathbf{U})\Psi = \mathbf{b}. \tag{22}$$

The Jacobi method converts this to the iterative equation,

$$\mathbf{D}\Psi^{k+1} = -(\mathbf{L}+\mathbf{U})\Psi^{k} + \mathbf{b}, \tag{23}$$

The next-simplest iterative method is Gauss-Seidel, which uses new values wherever they are available. This relies on sweeping across the grid in an ordered way. So if you start at $i, j = 0$ and then sweep across, increasing first i then j (or vice-versa), then it is possible to write an update formula,

$$\psi_{i,j}^{k+1} = \frac{1}{4}\left(\psi_{i+1,j}^{k} + \psi_{i-1,j}^{k+1} + \psi_{i,j+1}^{k} + \psi_{i,j-1}^{k+1} - \Delta x^2 \omega_{ij}\right), \quad i,j = 1,2,...,N, \tag{24}$$

Note now that superscript-$k+1$ appears on the right as well, for those values of ψ that already have a new estimate. The sweep will not have reached $i+1, j$ or $i, j+1$ yet, so we still use the old estimate k. The Gauss-Seidel method converges faster than the Jacobi method (but is hard to parallelise - you need not worry about this for your problem however). In matrix form, this method can be written,

$$(\mathbf{D}+\mathbf{L})\Psi^{k+1} = -\mathbf{U}\Psi^{k} + \mathbf{b}, \tag{25}$$

What do we mean by convergence of an iterative method? Quantitatively, what is a suitable measure to use for whether convergence has been reached?

Note that just because one value of $\psi_{i,j}$ has apparently converged, e.g. $\left|\psi_{i,j}^{k+1} - \psi_{i,j}^{k}\right| < \epsilon$ for some small value of ϵ, this may not be true for all values.

Finally, the most general method among this class is called the method of successive over-relaxation (SOR). It essentially combines Jacobi and Gauss-Seidel, and has a matrix representation,

$$(\mathbf{D}+r\mathbf{L})\Psi^{k+1} = -((1-r)\mathbf{L}+\mathbf{U})\Psi^{k} + \mathbf{b}, \tag{26}$$

where r is called a relaxation parameter. This recovers the Jacobi method for $r = 0$ and the Gauss-Seidel method for $r = 1$, but any choice of r is possible.

Derive an update formula for $\psi_{i,j}^{k+1}$ in terms of $\omega_{i,j}$ and appropriate values from the vectors Ψ^k and Ψ^{k+1}.

This basically concludes the preliminaries. You should now have a reasonable understanding of the physics, mathematics, and (initial) numerical features of the problem. In the next section you will implement this solution method and investigate its validity, accuracy, and performance - first with respect to a test case, and then with respect to our lid-driven cavity problem.

2.2 Implementation

It is now time to write your code.

Write a MATLAB code that estimates the streamfunction ψ from the Poisson equation, $\nabla^2\psi = \omega$, on a discretised square grid of size $N \times N$, where ω is considered known. Use the SOR method with a user-selectable over-relaxation parameter. Your code should output the number of SOR-iterations necessary for the solution to become converged.

The previous section gives you everything you need to know to do this, so we need not talk about how you would do it here. But when you write numerical code, you need a way to know if your code is right. This is called *verification*. To check this, we often use test cases; problems where we know analytically what the solution is, and we try to recover it.

We will use the following test case. The Poisson problem with a source function,

$$\omega(x, y) = 2\pi^2 \sin \pi x \sin \pi y, \tag{27}$$

with zero Dirichlet boundary conditions for Ψ, has solution

$$\psi(x, y) = -\sin \pi x \sin \pi y. \tag{28}$$

Your code will not reproduce the answer exactly. Intuitively you should understand

that if you choose N to be too small (e.g. 5), then your answer will be worse than if you choose a large N. In other words, N is a parameter that (along with other things) determines the accuracy of your code. In practice, you generally do not want your answer to depend on N, since N is not a physically relevant parameter. For sufficiently high N, for a well-written code, the answer becomes insensitive to N; at this point one can claim the code (or data produced by the code) is *grid-converged*. This is a separate idea from the convergence from iterating in e.g. the SOR method in the previous section.

Verify that your code approximately reproduces (28) when solving the Poisson equation (2) with $\omega(x, y)$ given by (27). Quantify the error using an appropriate measure. Then,

- For a fixed grid size N, plot the number of iterations required to converge as you vary the over-relaxation parameter r. Use this data to suggest an optimal value of r.
- Repeat this for a different grid size. Is the optimal value of r the same?
- Now fix r to a value, and vary the grid size N. Plot the error as a function of N on log-log axes. What can you deduce about the performance of your code?
- What is an acceptable error? Is it possible to get to zero error? What is an appropriate choice of N to use?
- With reference to section 2, how does the performance of your code depend on the way that the Laplacian operator has been discretised?

For your report: You are expected to produce the necessary plots and provide a critical discussion on what you have learned about the choices of numerical parameters r, N.

2.3 Solving the problem

It is time now to solve the real problem: finding the force on the driven lid. From before, for a given lid velocity U_0 and dynamic viscosity μ, the force F per unit

breadth of the lid is

$$F = \mu \int_0^{L_0} \left. \frac{\partial u_x}{\partial y} \right|_{y=L_0} dx. \tag{29}$$

You have written a solver for $\psi(x, y)$ on a discrete grid. To solve this problem, you have been supplied three different data files for $\omega(x, y)$, at different resolutions $N = 16, 32, 64$.

The structure of each file is as follows:

- The files are in a binary data format, recorded as single-precision floating point values.
- The data in its native form is just one long string of data, but can and should be arranged (with the code snippet below) as an $(N + 1) \times (N + 1)$ matrix.
- The first row records the x-coordinates x_i of the grid, the first column records the y-coordinates y_j of the grid.
- The remaining data constitutes $\omega_{ij} = \omega(x = x_i, y = y_j)$, depending on its location.

You can read the data into MATLAB as the square array described above with the following snippet,

```
fileid = fopen("omegaN64.dat");
dat = fread(fileid, "single");
N = 64;
dat = reshape(dat, [N+1, N+1]);
```

which you can proceed to use as you like.

Get a solution for $\psi(x, y)$ at three different resolutions $N = 16, 32, 64$ using the three different ω files which have been provided. Produce plots of the resulting contour map of ψ, for each resolution. Examine and interpret the results. Where are the largest velocities? Where are the zones of greatest shear?

Now to get the force, we need to estimate $\partial u_x/\partial y$ at the top of the box. Since $u_x = -\partial\psi/\partial y$, then clearly,

$$\frac{\partial u_x}{\partial y} = -\frac{\partial^2 \psi}{\partial y^2}. \tag{30}$$

But how to estimate this at the top of the box on a discretised grid, with $x = i\Delta x$, $y = j\Delta y$, with $i, j = 0, 1, 2, ..., N+1$? At the top of the box we have $j = N+1$. Obviously we will use a finite difference estimate. The obvious candidate is a backward difference evaluated at $j = N+1$,

$$\left.\frac{\partial^2 \psi}{\partial y^2}\right|_{y=L} \simeq \frac{\psi_{i,N+1} - 2\psi_{i,N} + \psi_{i,N-1}}{\Delta y^2} + O(\Delta y), \tag{31}$$

but this is not satisfactory because it is only first-order accurate. (Don't get confused between order of accuracy and order of the derivative, these are separate things.)

A first-order accurate estimate would not be inherently bad. Why should we insist on a second-order accurate estimate? Is there any point in going to a third-order accurate estimate?

Obviously we can get a second-order accurate estimate from a central difference formula, but this doesn't work at the top of the box because there is no $\psi_{i,N+2}$ point. So instead we use a second-order accurate backward difference to approximate the second derivative,

$$\left.\frac{\partial^2 \psi}{\partial y^2}\right|_{y=L} \simeq \frac{2\psi_{i,N+1} - 5\psi_{i,N} + 4\psi_{i,N-1} - \psi_{i,N-2}}{\Delta y^2} + O(\Delta y^2). \tag{32}$$

Find estimates of the force on the top of the box using (31), (32) for each resolution $N = 16, 32, 64$ and plot the results of each estimate as a function of resolution Δy. Use this data to produce a final estimate of the force on top of the box. *Hint: It should not simply be the second-order result corresponding to* $N = 64$!.

And that's it.

3 Your report

Having done the analysis in the previous sections, you will need to put your understanding into the report. I won't prescribe the exact structure of the report - that's up to you - but it should address the problems mentioned above. You may not necessarily include every boxed item that was written above - e.g. you may think that one point or another is trivial and needs no specific presentation - but in particular:

- A very brief description of the physical problem, its mathematical expression most relevant to your task, and how your numerical solution helps solve it. Don't just repeat all of what is in this document; the reader of your report doesn't need a tutorial, they need context. Think critically and use your judgment.

- A description of your numerical solver, how you constructed it, and its performance on the test problem. This might be a good place to talk about order of accuracy etc.

- The application of your solver to the actual problem, discussing its performance.

- Answer the question: "What is the force exerted by the fluid on the lid?" and give an uncertainty. Accuracy matters.

- Provide a discussion on the limitations of the model, and whether you think the accuracy that you report (e.g. in the previous point) is appropriate, or compromised by these limitations. The existence of limitations altogether doesn't necessarily mean the answer is wrong. Again, think critically and use your judgment.

The report is an opportunity for you to exercise your critical thinking skills, over and above the quantitative numerical exercise. It is easy to think of report-writing as a chore, but it can instead be an interesting discussion on a topic.

Lastly I care about the quality of the report, not how you wrote it. If you use ChatGPT for the report (or code for that matter) I may not be able to tell, but if you do use it uncritically this could just mean that I couldn't distinguish it from a *bad report written by a human*. Instead if you want to use it, please think carefully about your inputs, tune or rewrite the outputs, and/or iterate on the results - this will give you the best chance to end up with a good report.