Analysing the mathematical model of Sea-Breeze Flow

Exploring possibilities for the mass density function for a mathematical model of Sea-Breeze flow

Guo Chuen Liu

TU Delft

Analysing the mathematical model of Sea-Breeze Flow

Exploring possibilities for the mass density function for a mathematical model of Sea-Breeze flow

by

Guo Chuen Liu

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on July 9, 2025 at 14:00.

Student number:	5312248	
Project duration:	April 24, 2025 – July 9, 2025	
Thesis committee:	Dr. K. Marynets,	TU Delft, supervisor
	Dr. A. Palha da Silva Clérigo,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

I would like to thank Kateryna Marynets for her support during the project, as well as for the helpful feedback she provided for my drafts. I would also like to thank Artur Palha da Silva Clérigo for being part of my graduation committee and for giving the course Numerical Methods 2.

Guo Chuen Liu
Delft, July 2025

Layman's Summary

Sea breezes are atmospheric flows that move air from the sea towards the land and primarily occur on the shore. They form due to temperature differences that develop during the day, as the sea heats up more slowly than land. The goal of this project is to study the solutions to a mathematical model describing sea breeze flows in the Gulf of Carpentaria (Australia) and Calgary (Canada) under various physical conditions. The model uses an ordinary differential equation to describe the horizontal velocity of the flow at different altitudes. Using numerical methods, we computed the horizontal velocity of the flow for three types of external forcing. For the Gulf of Carpentaria, we found that in the absence of any forcing, sea breezes do not occur. Secondly, for a constant force, the entire flow moves at a constant speed everywhere. Lastly, with a forcing term that increases with height, we found that the velocity also increases with height. Similarly, we computed the horizontal velocity of breezes in Calgary. Because Calgary lies in the Northern Hemisphere, we found that the winds flow in the opposite direction compared to those in the Gulf of Carpentaria. This difference is caused by the Coriolis force, which results from Earth's rotation. The Coriolis force causes winds to be deflected to the right in the Northern Hemisphere and to the left in the Southern Hemisphere, explaining the opposite wind directions.

Summary

Sea breezes are low-level atmospheric flows that move from the sea to land. They typically form during warm days when the temperature difference between the land and sea surfaces becomes large enough. This temperature difference occurs because seawater, having a higher heat capacity, heats up more slowly than land. In this thesis, we study a nondimensional mathematical model that describes the horizontal velocity of sea breezes in two regions: the Gulf of Carpentaria (Australia) and Calgary (Canada). The model was derived in [4] by writing the Navier-Stokes equations in rotating spherical coordinates. After nondimensionalisation and assuming a specific form of the solution, they obtained the nondimensional model. Furthermore, the model was complemented by two homogeneous Dirichlet boundary conditions, representing the no-slip condition at Earth's surface and the presence of a thermal inversion layer. In [11], their work was extended by studying the corresponding Sturm-Liouville problem for various mass density functions to determine the existence and uniqueness of the solutions to the model.

Building on the work in [11], we analysed the solvability of the model and determined the numerical solutions for two physically relevant mass density functions. These functions were chosen to ensure that zero is not an eigenvalue. The Fredholm alternative then implies that the solution is unique. On the other hand, if zero were an eigenvalue, then the Fredholm alternative implies that we could have an infinite number of solutions, which is physically unrealistic.

For the first mass density function, the eigenvalue problem was written as the hypergeometric equation. For the Gulf of Carpentaria (where $\beta > 0$), we proved analytically that zero is not an eigenvalue. However, since the parameter β is negative for the Calgary region, we were unable to prove this analytically. Instead, we demonstrated numerically that the eigenvalues closest to zero were $\lambda_{807} = -828$ and $\lambda_{808} = 790$, confirming that zero is not an eigenvalue. Using the finite difference method, we then computed the velocity profiles for three different forcing functions. We first considered the breezes in the Gulf of Carpentaria. In the first case we assumed zero forcing, the result showed that in the absence of any forcing, sea breezes do not occur. For the second case we assumed a non-zero constant forcing, resulting in a uniform velocity profile. Lastly, with an increasing quadratic forcing function, the velocity increased quadratically with height, reaching its maximum just below the thermal inversion layer. Similarly, we computed the velocity profiles for the Calgary region. From these computations, we found that the breezes were flowing in the opposite direction and with higher velocities. These differences arise due to the Coriolis force, which forms due to Earth's rotation. This force deflects winds to the right in the Northern Hemisphere and to the left in the Southern Hemisphere. Furthermore, the Coriolis force is zero at the equator and grows with latitude, which explains the higher velocities.

Finally, we considered the model for the second mass density function, which includes the weak effect the boundaries have on the flow. For the Gulf of Carpentaria it was again proven analytically that zero was not an eigenvalue, while for the Calgary region, the eigenvalues closest to zero were $\lambda_{501} \approx -351$ and $\lambda_{502} \approx 3030$. Similar to the first mass density function, we used the finite difference method to compute the velocity profiles. The results showed that the profiles were identical to those obtained with the first mass density function. This similarity occurs because the two mass density function exhibit similar parabolic behaviour. Moreover, the same forcing functions were used in both cases.

Contents

1

Introduction

Sea breeze is a low-level atmospheric flow that moves from the sea to land [11]. Sea breezes typically form during the day when the temperature difference between the land and sea surfaces becomes large enough [6]. This usually happens during warm periods, since the sun heats the sea much more slowly than the land. This difference in heating is due to the higher heat capacity of sea water ($3.93 \cdot 10^3$ J kg^{-1} K^{-1}) compared to the much lower heat capacity of sand ($0.80 \cdot 10^3$ J kg^{-1} K^{-1}) [3] (for more information about other factors that are relevant to this process, we refer the reader to [1, 13]). As a result, the warm ground heats the air above it, making it warmer than the air over the sea surface. Consequently, the warm air rises, creating a low-pressure area called a "thermal low" directly above the ground. This low-pressure area eventually leads to the formation of additional high- and low- pressure regions, as can be seen in Figure 1.1. The resulting pressure differences between these regions give rise to the sea breeze flow and the return flow, ultimately leading towards the formation of the sea breeze circulation (see Figure 1.1) [13].

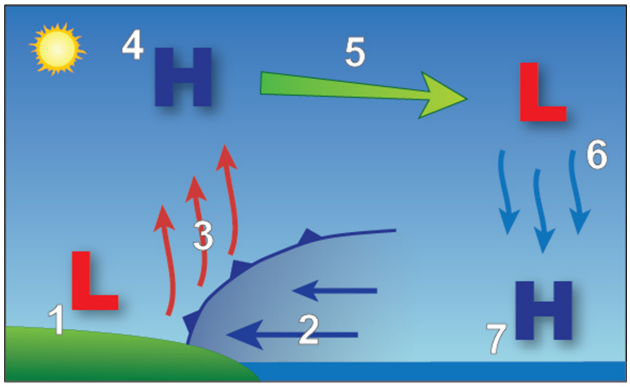


Figure 1.1: The sea breeze circulation, where L and H stand for low and high pressure areas respectively. The "thermal low" corresponds to (1), the warm return flow to (5) and the (cool) sea-breeze flow to (2) (source: [13]).

To better understand these flows, researchers extensively use their mathematical formulations involving nonlinear partial differential equations. In particular, the authors of [4] studied the mechanism behind the morning glory phenomenon, a cloud formation associated with the flow of nonlinear waves. Based on the Navier-Stokes equations written in spherical coordinates, they derived a nondimensional mathematical model to describe the behaviour of the phenomenon [4]. By assuming a specific form of the solution, they were able to establish a model describing the horizontal velocity of sea breeze flows. In [11], their work was extended by studying the corresponding Sturm-Liouville boundary value problem for various mass density profiles. By applying the Sturm-Liouville theory, the author proved the existence and uniqueness of the solutions to the original problem.

For this project, we build on the work in [11] by exploring other physically realistic possibilities for the mass density functions. Specifically, we focus on functions for which zero is not an eigenvalue, as a zero eigenvalue

could imply that the solution is not unique. This would mean that the flow could be described by multiple distinct velocity profiles, which is not physically realistic. By considering various mass density functions, we aim to determine the eigenvalues of the Sturm-Liouville problem corresponding to the model of sea breeze flows for the Gulf of Carpentaria and the Calgary region. Then by using numerical methods we compute the solution to the model with various forcing functions, followed by a physical interpretation of the obtained results.

The thesis is structured in the following way. In Chapter 2, we study the mathematical model of sea breeze that was derived in [4]. In Chapter 3, we analyse the Sturm-Liouville problem corresponding to the model for various mass density functions. Based on these results, we study the solutions to the original model. Finally, in Chapter 4, we present our conclusions and suggest possible extensions for future work.

2

Model of Sea Breeze flow

This chapter provides an overview of the model of sea breeze flows and the relevant boundary conditions that were derived in [4]. Section 2.1 focuses on the nondimensional model of breeze-like flows. In Section 2.2 the simplified model for the flows in the Gulf of Carpentaria will be examined. Finally, in Section 2.3, the corresponding Sturm-Liouville problem of the simplified model will be studied.

2.1. The nondimensional model

To model atmospheric flows, the authors in [4] began with the Navier-Stokes equations formulated in rotating spherical coordinates. They then nondimensionalised the resulting equations, which yielded a model for non-linear wave propagation in the atmosphere. By assuming a specific form of the solution (the exact form of the solution is provided in [4]), they derived a nondimensional model of sea breeze flow. This model describes the horizontal velocity of the flow, denoted by $V_0(z,\Phi)$, and is given by:

$$\rho\sigma S V_0 + \frac{1}{R_e}\frac{\partial}{\partial z}\left(m(z)\frac{\partial V_0}{\partial z}\right) = -\left\{\cos^2(\alpha) + \frac{\sin^2(\alpha)}{C}\right\}K(z,\Phi), \qquad 0 < z < z_0, \tag{2.1}$$

where from [11]

- z corresponds to the height inside the air flow;
- $\rho_0(z)$ is the density function;
- S, α, C, $\sigma = \frac{2(\sin^2(\alpha)+C\cos^2(\alpha)}{(1-C)\sin(\alpha)\cos(\alpha)}$, are the characteristics of the flow in the specific region,
- $R_e \approx 10^5$ is the Reynolds number;
- $m(z)$ is the viscosity function;
- Φ is a parameter, corresponding to the direction of flow propagation and
- $K(z,\Phi)$ is the forcing term in the model.

However, Equation (2.1) alone is not enough to fully describe the sea breeze, as it does not specify the behaviour of the flow at the boundaries. To complete the model, Equation (2.1) is complemented by two boundary conditions. The first occurs at the ground ($z = 0$), where due to friction of the airflow with the surface, the horizontal velocity of the sea breeze must be zero [4]

$$V_0(0,\Phi) = 0.$$

This condition is called the no-slip condition at the Earth's surface [4].

The second boundary condition occurs due to the presence of a thermal inversion layer at $z = z_0$ [4]. A thermal inversion layer is a region in the atmosphere where the temperature increases with height instead of decreasing. This thermal inversion layer corresponds to the warm return flow (see (5) in Figure 1.1) in the sea breeze circulation, which starts at z_0. As a result, the return flow and the sea breeze collide at $z = z_0$, cancelling the horizontal velocity of the breeze-like flow. Therefore, at the height z_0, the horizontal velocity of the sea breeze must also be zero [4]

$$V_0(z_0,\Phi) = 0.$$

2.2. Simplified nondimensional model

For the sea breeze flow in the Gulf of Carpentaria, Australia (see Figure 2.1), the parameters in Equation (2.1) take the following values [4, 11]:

$$C \approx 0.97,\ S \approx -0.24,\ \sigma \approx 133 \text{ and } \alpha = \frac{5\pi}{4}.$$

Using these values, the model can be simplified to describe the flow propagating in the south-west direction in the Gulf of Carpentaria [4]. Thus, Equation (2.1) is written as:

$$\beta V_0 - \frac{\partial}{\partial s}\Big(\hat{m}(s)\frac{\partial V_0}{\partial s}\Big) = k_0(s,\Phi), \qquad 0 < s < 1, \tag{2.2}$$

with new parameters and variables given by [4]:

$$\beta = -\sigma S R_e > 0, \qquad s = \frac{1}{\int_0^{z_0}\rho_0(\xi)d\xi}\int_0^z \rho_0(\xi)d\xi,$$

$$\hat{m}(s) = \frac{\rho_0(s)m(s)}{(\int_0^{z_0}\rho_0(\xi)d\xi)^2}, \qquad k_0(s,\Phi) = R_e\Big\{\cos^2(\alpha) + \frac{\sin^2(\alpha)}{C}\Big\}\frac{K(s,\Phi)}{\rho_0(s)},$$

Figure 2.1: The Gulf of Carpentaria lies in the northern part of Australia, and in the figure, north corresponds to the upward direction (source: [14]).

where k_0 is the scaled forcing function.
Next, for the simplified model in Equation (2.2), it is assumed that the $\rho_0(z)$ and $K(z,\Phi)$ are continuous functions (with Φ being a parameter), while $m(z)$ is continuously differentiable [4, 11]. Furthermore, for every fixed Φ (i.e., the direction of flow propagation is fixed), the boundary conditions for the simplified model (2.2), can be written as homogeneous Dirichlet boundary conditions [4, 11]:

$$V_0(0) = 0,\ V_0(1) = 0. \tag{2.3}$$

Thus the model, for which we are exploring possibilities of the mass density function $\hat{m}(s)$ is given by:

$$\begin{cases} \beta V_0 - \frac{d}{\partial s}\Big(\hat{m}(s)\frac{dV_0}{ds}\Big) = k_0(s), & 0 < s < 1, \\ V_0(0) = 0, \\ V_0(1) = 0. \end{cases} \tag{2.4}$$

2.3. Corresponding Sturm-Liouville problem

The Fredholm alternative [8] characterises the solvability of a (nonhomogeneous) boundary value problem depending on the eigenvalues of the corresponding Sturm-Liouville problem

$$L(u) = \lambda u(x).$$

Theorem 1. *[8] Consider the nonhomogeneous differential equation*

$$L(u) = f(x), \quad a < x < b,$$

subject to homogeneous boundary conditions. Then

- *If $\lambda = 0$ is not an eigenvalue, then the nonhomogeneous problem has a unique solution.*
- *On the other hand, if $\lambda = 0$ is an eigenvalue, then the nonhomogeneous problem has either no solution or infinitely many solutions.*

If zero is an eigenvalue, then the problem has infinitely many solutions if the orthogonality condition

$$\int_a^b f(x)\phi_h(x)\,dx = 0$$

is satisfied, where $\phi_h(x)$ corresponds to the homogeneous solution.

To draw conclusions about the existence and uniqueness of solutions to Equation (2.4), the author in [11] studied the corresponding Sturm-Liouville problem given by:

$$\begin{cases} \beta V_0 - \frac{d}{ds}\left(\hat{m}(s)\frac{dV_0}{ds}\right) = \lambda V_0(s), & 0 < s < 1, \\ V_0(0) = 0, \\ V_0(1) = 0, \end{cases} \tag{2.5}$$

where $\hat{m}(s)$ does not change sign [11].
However, since the mass density function $\hat{m}(s)$ is unknown, the author used Sturm-Liouville theory [2, 7, 10] to derive suitable forms of $\hat{m}(s)$ that transform Equation (2.5) into known Sturm-Liouville problems, such as the Legendre equation. In [4], it was shown that if $\rho_0(z)$ and $K(z,\Phi)$ are continuous, and $m(z)$ is continuously differentiable, then the self-adjoint unbounded linear operator

$$L = \beta - \frac{\partial}{\partial s}(\hat{m}(s)\frac{\partial}{\partial s}),$$

acting in $L^2(0,1)$ has discrete eigenvalues. Furthermore, the corresponding eigenfunctions form an orthonormal basis. Using these eigenfunctions, it was also shown that all eigenvalues are positive when $\beta > 0$, since $\lambda_1 > \beta$, where λ_1 is the smallest eigenvalue. By applying the Fredholm alternative, the authors concluded that the solutions to the model (2.1) exist and are unique. However, in some regions β may be negative, which could cause zero to be an eigenvalue. An example of such a region is the Calgary region, Canada (see Figure 2.2), where $C \approx 0.62$, $S \approx 0.77$, $\sigma \approx 8.5$ (so $\beta < 0$) with landlocked breezes flowing the in North-West direction ($\alpha = \frac{\pi}{4}$) [4]. In these cases, the Fredholm alternative is used to derive the orthogonality condition on the right-hand side of the model (2.4) to ensure the existence of solutions [4, 11].

Figure 2.2: Geographical location of the Calgary region (marked in red), which is a landlocked region in southern Canada (source: Google Maps).

In [11], the author considered model (2.4) with a quadratic mass density function $\hat{m}(s)$. Specifically, for $\hat{m}(s) = (as+b)^2$, the corresponding Sturm-Liouville problem becomes

$$\begin{cases} -(as+b)^2 V_0''(s) - 2a(as+b)V_0'(s) + \beta V_0(s) = \lambda V_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0. \end{cases} \tag{2.6}$$

This equation is related to the Legendre equation, meaning that it can be reduced to a differential equation with constant coefficients. To obtain non-trivial eigenfunctions, the author showed that the eigenvalues λ must satisfy the inequality $\sqrt{a^2 + 4(\lambda - \beta)} < 0$. By solving the differential equation, the author obtained explicit expressions for the eigenfunctions and eigenvalues. Based on the explicit formula for the eigenvalues, it was concluded that all eigenvalues are positive when $\beta \geq 0$. In this case, the author used the Fredholm alternative to conclude that the solution to model (2.4) exists and is unique. However, when $\beta < 0$, the author showed that zero could be an eigenvalue. In that case, the author derived an orthogonality condition[1] for $k_0(s)$ to ensure that the solution is unique.

[1] In [11], the author uses a different orthogonality condition than in Theorem 1. If zero is an eigenvalue and the forcing function $k_0(s)$ is orthogonal to the eigenfunctions, then the solution exists and is **unique**.

In [16], the authors built upon the work in [11] by investigating the differential equation with constant coefficients (as derived in [11]) with the same mass density function for both a linear and a general nonlinear forcing term $k_0(s)$. For the linear case, they considered $k_0(s) = mV(s) + n$, where m and n are nonzero constants. For this forcing term, they derived explicit solutions and showed that the differential equation exhibits semi-Ulam-Hyers stability. They then considered the nonlinear case, where they used Green's functions and the fixed-point theorem to prove that a solution to the model exists and is unique.
Motivated by the mentioned above results, we will aim to explore further possibilities for the mass density function that would lead to unique solutions to model (2.4).

3

Methods and results

This chapter analyses the Sturm-Liouville problem (2.5) for two different mass density functions, denoted by $\hat{m}(s)$. Based on this analysis, we draw conclusions about the existence and uniqueness of solutions to model (2.4) for both the Gulf of Carpentaria and the Calgary region. For each mass density function we also numerically determine the solution to the model for both regions, followed by a physical interpretation of the results. Section 3.1 analyses the eigenvalue problem, which is reformulated as the hypergeometric equation. Lastly, in Section 3.2, we study the eigenvalue problem for a physically relevant mass density function.

3.1. Hypergeometric equation

By introducing the mass density function as $\hat{m}(s) = s - s^2$, model (2.4) is written as:

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left((s-s^2)\frac{dV_0}{ds}\right) = k_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0. \end{cases} \tag{3.1}$$

The corresponding Sturm-Liouville problem (2.5) is then given by:

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left((s-s^2)\frac{dV_0}{ds}\right) = \lambda V_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0. \end{cases} \tag{3.2}$$

This equation is equivalent to the hypergeometric equation[1] with parameters $\alpha = 0$ and $\gamma = 0$ [2]. To see this, we rewrite Equation (3.2) as:

$$-\frac{d}{ds}\left(s(1-s)\frac{dV_0}{ds}\right) = \mu V_0(s), \tag{3.3}$$

where

$$\mu = \lambda - \beta. \tag{3.4}$$

This can be compared to the hypergeometric equation[2] [2], which has the same homogeneous boundary conditions

$$-(s^{\alpha+1}(1-s)^{\gamma+1}V_0'(s))' = \mu s^{\alpha}(1-s)^{\gamma}V_0(s) \quad 0 < s < 1. \tag{3.5}$$

From this comparison, it follows that the eigenvalues μ of the hypergeometric equation satisfy Equation (3.4). Consequently, the eigenvalues of the Sturm-Liouville problem (3.2) are expressed as

$$\lambda_n = \mu_n + \beta \text{ for n} \in \mathbb{N}.$$

However, since the spectrum and eigenfunctions of the hypergeometric equation have not been fully explored, an explicit formula for the eigenvalues μ and eigenfunctions are not (yet) known [2]. Therefore, we

[1] In [2], the authors use the parameters α and β. However, since the model already uses β, we replace it with γ in the hypergeometric equation to avoid confusion.

[2] The eigenvalue of this equation is given by μ, while the eigenvalue of the model is given by λ.

use the finite difference method to numerically compute the eigenvalues μ and eigenfunctions of the hypergeometric equation. We begin by discretising the interval $[0,1]$ using N interior points as illustrated in Figure 3.1.

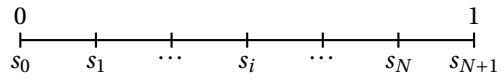


Figure 3.1: Discretisation of the domain $[0,1]$ into discrete points s_i.

We define the grid points as $s_i = ih$, for $0 \le i \le N+1$ with stepsize $h = \frac{1}{N+1}$, such that $V_i = V(s_i)$, $\hat{m}_i = \hat{m}(s_i) = s_i - s_i^2$. For simplicity we will write $V_0(s) = V(s)$ to avoid any confusion with the indices. Let y_i be the numerical approximation of $V(s_i) = V_i$, then for $1 < i < N+1$ the discretised model can be written as:

$$\begin{cases} -\frac{d}{ds}\left((s-s^2)\frac{dy}{ds}\right)\Big|_i = \mu y_i, \quad 0 < s < 1 \\ y_0 = y_{N+1} = 0. \end{cases} \tag{3.6}$$

To solve this equation, we apply central differences to **approximate** the derivates as shown in Equation (3.7). Note that the local truncation error of the central difference approximation is of order $\mathcal{O}(h^2)$ (see Appendix A). This implies that the numerical method is consistent [15], meaning that the local truncation error tends to zero as $h \to 0$.
We define $\hat{m}_{i+\frac{1}{2}} = \hat{m}(\frac{s_i+s_{i+1}}{2})$ and $\hat{m}_{i-\frac{1}{2}} = \hat{m}(\frac{s_i+s_{i-1}}{2})$ then applying central differences gives

$$\begin{aligned} &-\frac{\hat{m}_{i+\frac{1}{2}}\frac{dy}{ds}|_{i+\frac{1}{2}} - \hat{m}_{i-\frac{1}{2}}\frac{dy}{ds}|_{i-\frac{1}{2}}}{2\frac{h}{2}} = \mu y_i \\ &\text{Apply the central difference to the first derivatives.} \\ &\frac{-\hat{m}_{i+\frac{1}{2}}y_{i+1} + (\hat{m}_{i+\frac{1}{2}} + \hat{m}_{i-\frac{1}{2}})y_i - \hat{m}_{i-\frac{1}{2}}y_{i-1}}{h^2} = \mu y_i, \quad \text{for } 1 < i < N. \end{aligned} \tag{3.7}$$

For $i = 1$ we obtain:

$$\begin{aligned} &\frac{-\hat{m}_{\frac{3}{2}}y_2 + (\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}})y_1 - \hat{m}_{\frac{1}{2}}y_0}{h^2} = \mu y_1 \\ &\text{Since } y_0 = 0 \text{ we find:} \\ &\frac{-\hat{m}_{\frac{3}{2}}y_2 + (\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}})y_1}{h^2} = \mu y_1. \end{aligned} \tag{3.8}$$

Similarly for $i = N$ we get:

$$\frac{(\hat{m}_{N+\frac{1}{2}} + \hat{m}_{N-\frac{1}{2}})y_N - \hat{m}_{N-\frac{1}{2}}y_{N-1}}{h^2} = \mu y_N. \tag{3.9}$$

Thus, from the finite difference method we obtain the system $A\mathbf{y} = \mu\mathbf{y}$, where A (an $N \times N$ matrix) and $\mathbf{y}$ are given by:

$$A = \begin{bmatrix} \frac{\hat{m}_{\frac{3}{2}}+\hat{m}_{\frac{1}{2}}}{h^2} & -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & 0 & 0 & 0 & 0 \\ -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & \frac{\hat{m}_{\frac{5}{2}}+\hat{m}_{\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & 0 & 0 & 0 \\ 0 & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & \frac{\hat{m}_{\frac{7}{2}}+\hat{m}_{\frac{5}{2}}}{h^2} & -\frac{\hat{m}_{\frac{7}{2}}}{h^2} & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{3}{2}}}{h^2} & \frac{\hat{m}_{N-\frac{1}{2}}+\hat{m}_{N-\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} \\ 0 & 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} & \frac{\hat{m}_{N+\frac{1}{2}}+\hat{m}_{N-\frac{1}{2}}}{h^2} \end{bmatrix}, \tag{3.10}$$

$$\mathbf{y}^T = \begin{bmatrix} y_1 & y_2 & \dots & y_N \end{bmatrix}^T. \tag{3.11}$$

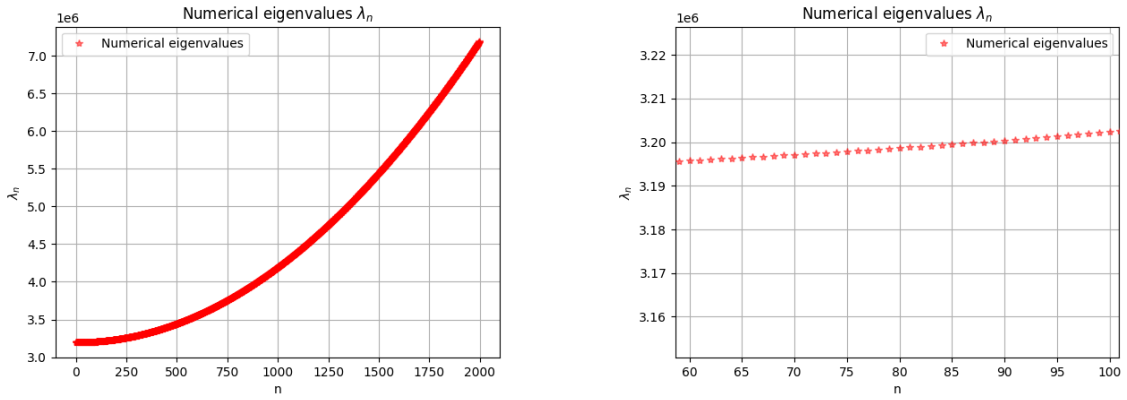
By solving this eigenvalue problem and using that $\lambda_n = \mu_n + \beta$, we obtain the eigenvalues λ and eigenvectors of Equation (3.2). Furthermore, for the Gulf of Carpentaria we have $\beta_{\text{Carpentaria}} \approx 3.192 \cdot 10^6$ and for the Calgary

region we have $\beta_{\text{Calgary}} \approx -6.545 \cdot 10^5$. It should be noted that the nth eigenvector represents the values of the nth eigenfunction at discrete points. The corresponding Python implementation is provided in Appendix D.1.

With the approximations for the eigenvalues λ_n, we examine the eigenvalue problem (3.2) for two specific regions: the Gulf of Carpentaria and the Calgary region. It is important to note that the eigenfunctions for both equations are the same, since the only difference is that the eigenvalues are shifted by β.

3.1.1. Gulf of Carpentaria

From [4], it follows that in the Gulf of Carpentaria, we have $C \approx 0.97$, $\alpha = \frac{5\pi}{4}$, $\sigma \approx 133$, $S \approx -0.24$, and therefore $\beta_{\text{Carpentaria}} = -\sigma S R_e > 0$. Using $N = 2000$ points in the finite difference method (see Section 3.1), we compute the first 2000 eigenvalues of the Sturm-Liouville problem (3.2). By the Gershgorin circle theorem [15] (see Appendix C), we find that all eigenvalues are positive and that the smallest eigenvalue satisfies $\lambda_1 \geq \beta_{\text{Carpentaria}}$. This theoretical lower bound is consistent with our numerical results as shown in Figure 3.2, where the smallest eigenvalue is $\lambda_1 = 3192000.261076134$. Since zero is not an eigenvalue, the Fredholm alternative [8] implies that the solution to model (3.1) for the Gulf of Carpentaria exists and is unique for all forcing functions $k_0(s)$.



(a) (b)

Figure 3.2: The numerical eigenvalues of the Sturm-Liouville problem (3.2) for the Gulf of Carpentaria. (b) Detailed view of the eigenvalues, showing that the eigenvalues are discrete.

Theorem 2. *The solution to the boundary value problem in the Gulf of Carpentaria*

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left((s-s^2)\frac{dV_0}{ds}\right) = k_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0, \end{cases}$$

exists and is unique for all forcing functions $k_0(s)$, provided that $\beta = \beta_{Carpentaria} \approx 3192000$.

Proof. Suppose that $\lambda = 0$ is an eigenvalue, then the eigenvalue problem (3.2) is written as:

$$\beta V_0(s) - \frac{d}{ds}\left((s-s^2)\frac{dV_0(s)}{ds}\right) = 0.$$

Multiplying both sides by $V_0(s)$ and integrating gives:

$$\int_0^1 \beta V_0^2(s)\,ds - \int_0^1 \frac{d}{ds}\left((s-s^2)\frac{dV_0(s)}{ds}\right)V_0(s)\,ds = 0$$

$$\beta \int_0^1 V_0^2(s)\,ds - \left[(s-s^2)V_0'(s)V_0(s)\right]_0^1 + \int_0^1 (s-s^2)(V_0'(s))^2\,ds = 0.$$

Using the homogeneous Dirichlet boundary conditions we find:

$$\beta \int_0^1 V_0^2(s)\,ds - 0 + \int_0^1 (s-s^2)(V_0'(s))^2\,ds = 0$$

$$\beta \int_0^1 V_0^2(s)\,ds = -\int_0^1 (s-s^2)(V_0'(s))^2\,ds.$$

Since $(s-s^2) \geq 0$ for $0 \leq s \leq 1$, and $(V_0'(s))^2 \geq 0$, it follows that the right-hand side is non-positive, which implies that we have:

$$\beta \int_0^1 V_0^2(s)\, ds \leq 0. \tag{3.12}$$

Using Equation (3.12) and that $V_0^2(s) \geq 0$, we find that $0 \leq \int_0^1 V_0^2(s) \leq 0$. But this means that $\int_0^1 V_0^2(s) = 0$, which in turn means that $V_0(s) = 0$. Thus for $\lambda = 0$, we obtain the trivial solution, contradicting our assumption that 0 was an eigenvalue. By applying the Fredholm alternative [8], we conclude that the solution to Equation (3.2) exists and is unique for all forcing functions $k_0(s)$. □

We now consider the solutions to model (3.1) with the mass density function $\hat{m}(s) = s - s^2$. It is important to note that the mass density function $\hat{m}$ is not the same as the density function ρ_0. While ρ_0 represents mass per unit volume, the function $\hat{m}$ describes the mass flux, defined as the amount of mass flowing per unit time through a unit surface area. As shown in Figure 3.3, the mass density function reaches its maximum at the midpoint and decreases towards the boundaries. This behaviour is similar to the velocity profile of laminar flow in a closed pipe, where the velocity of the fluid is maximal in the centre and minimal near the walls [9].

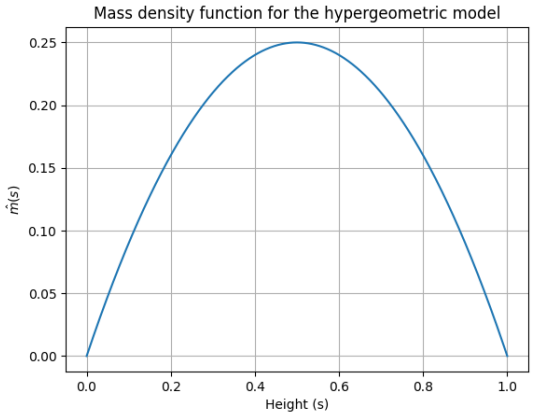


Figure 3.3: The mass density function used in model (3.1).

To study the influence of the forcing term $k_0(s)$ on the velocity of sea breeze flows, we first solve the homogeneous model equation using the finite difference method with $N = 2000$ points (see Appendix B.2), which gives the system

$$B\mathbf{y} = \mathbf{b}.$$

From Appendix A, it follows that the local truncation error of the finite difference method is of order $\mathcal{O}(h^2)$, which implies that the method is consistent. The resulting symmetric discretisation matrix B is invertible, because its columns span $\mathbb{R}^N$ (this is proven by the python implementation provided in Appendix D.2). Furthermore, by applying the Gershgorin circle theorem [15] (see Appendix C), we find that the smallest absolute eigenvalue[3] satisfies $\lambda_{\min} \geq \beta_{\text{Carpentaria}}$. This means that we have[4]

$$||B^{-1}|| = \frac{1}{|\lambda|_{\min}} = \frac{1}{\min\limits_{1\leq i\leq N} |\lambda_i|} \leq C,$$

where C is a constant (in this case we have that $C = \frac{1}{\beta_{\text{Carpentaria}}}$) independent of h. Then [15] implies that the method is stable and since it is also consistent, we conclude that the method is convergent.

The resulting velocity profile of the sea breeze, in the absence of a forcing term, is plotted in Figure 3.4.

[3] In this case all eigenvalues are positive.

[4] We use the natural matrix norm [15], which is defined as $||B|| = \max\limits_{||\mathbf{w}||=1} ||B\mathbf{w}||$. For symmetric matrices we have $||B|| = |\lambda|_{\max} = \max\limits_{1\leq i\leq N} |\lambda_i|$ and $||B^{-1}|| = \frac{1}{|\lambda|_{\min}} = \frac{1}{\min\limits_{1\leq i\leq N} |\lambda_i|}$.

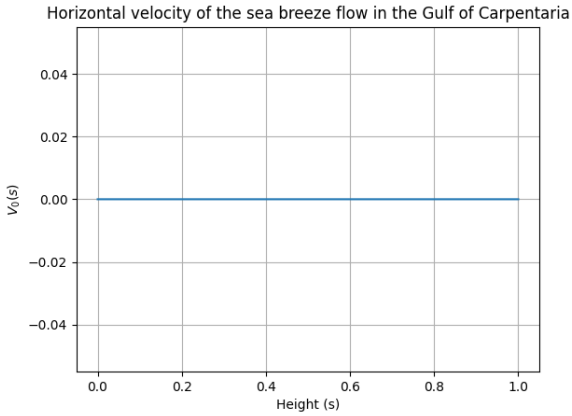


Figure 3.4: The horizontal velocity of the sea breeze in the Gulf of Carpentaria when no forcing terms are present.

From the figure, it is clear that the horizontal velocity of the sea breeze flow is zero everywhere. Thus, in the absence of a forcing term, sea breezes do not occur in the Gulf of Carpentaria. Physically, this result is to be expected, since without a forcing term, the wind forces responsible for generating sea breezes are absent.

Next, we assume that $k_0(s)$ is non-zero. Since

$$k_0(s) = R_e\Big\{\cos^2(\alpha) + \frac{\sin^2(\alpha)}{C}\Big\}\frac{K(s)}{\rho_0(s)}, \tag{3.13}$$

we need to specify both the forcing function $K(s)$ and the density function $\rho_0(s)$. To simplify for our calculations, we assume that the term $\frac{K(s)}{\rho_0(s)}$ represents wind forces. As an initial guess, we take $\frac{K(s)}{\rho_0(s)} = 1$, which means that forcing function k_0 is constant and has the form

$$k_0(s) = R_e\Big(\cos^2(\alpha) + \frac{\sin^2(\alpha)}{C}\Big). \tag{3.14}$$

The solution is computed using the finite difference method with $N = 2000$ points (see Appendix B.2), and the result is plotted in Figure 3.5.

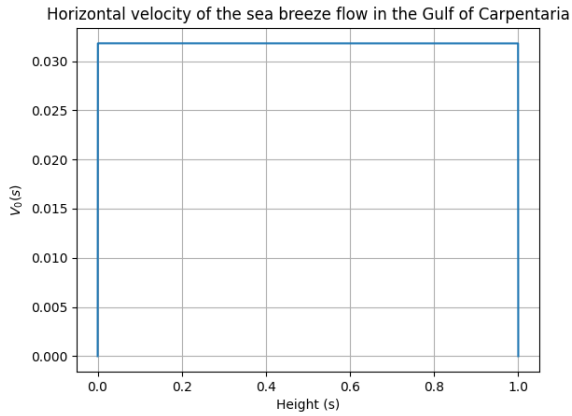


Figure 3.5: The horizontal velocity profile of the sea breeze in the Gulf of Carpentaria with a constant forcing function.

From the results, we see that in the presence of constant wind forcing, the horizontal velocity of the sea breeze remains constant in the interval $(0,1)$. It drops only zero at the boundaries due to the no-slip condition and the thermal inversion layer. Physically, this does make sense, if the same force is applied everywhere to the mass, then the whole mass would eventually move with the same speed.

Lastly, we assume that $\frac{K(s)}{\rho_0(s)} = 5s^2 + s + 1$, which means that the forcing function k_0 is an increasing quadratic function with the following form

$$k_0(s) = R_e\Big(\cos^2(\alpha) + \frac{\sin^2(\alpha)}{C}\Big)(5s^2 + s + 1). \tag{3.15}$$

Using the finite difference method with $N = 2000$ points (see Appendix B.2), the resulting velocity profile is plotted in Figure 3.6.

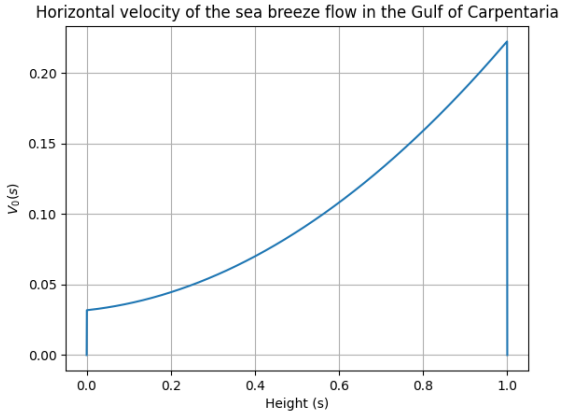
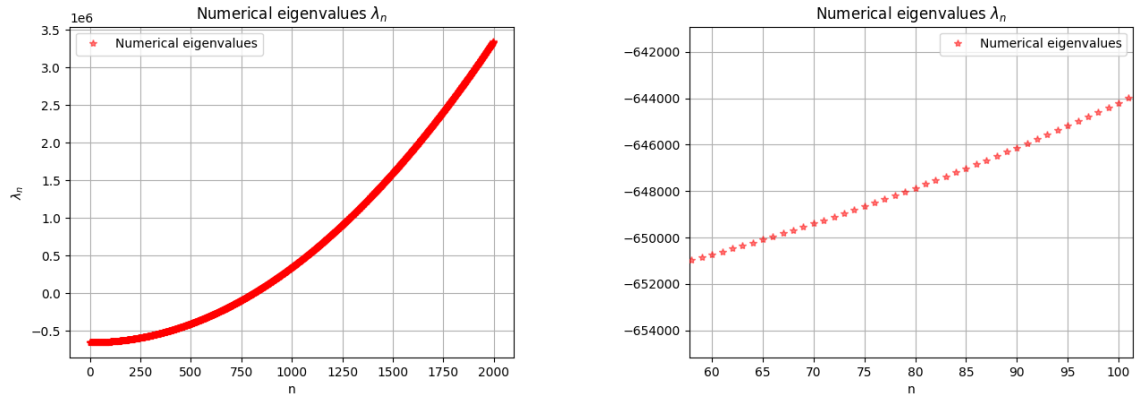


Figure 3.6: The horizontal velocity of the sea breeze in the Gulf of Carpentaria for an increasing quadratic forcing function.

In Figure 3.6, we observe that the velocity of the sea breeze increases quadratically with height under the influence of a quadratic forcing function. Due to the no-slip condition and the presence of a thermal inversion layer, the velocity is zero at both boundaries. There are two reasons for the behaviour of this solution. The first reason is that the forcing function k_0 starts small and increases with height. Wind forces near the ground are small, but they grow quadratically as the height increases. Since k_0 is never zero, wind forces are always present, which causes a jump in the velocity immediately after $s = 0$. The second reason is that the air density is not constant, but decreases with height. Near the ground we have the heaviest air, meaning that the weakest wind forces must push the heaviest mass, resulting in the slowest velocities for the sea breeze flow. On the other hand, higher up in the atmosphere, we have the lightest air being pushed by the strongest wind forces. As a result, the sea breeze flow reaches its maximal velocity just below the thermal inversion layer.

3.1.2. Calgary region

From [4], it follows that in the Calgary region, we have $C \approx 0.62$, $\alpha = \frac{\pi}{4}$, $\sigma \approx 8.5$, $S \approx 0.77$ and this means that the parameter β is negative. In the same way as for the Gulf of Carpentaria (see Section 3.1.1), the first 2000 eigenvalues of the Sturm-Liouville problem (3.2) with $\beta < 0$ are plotted in Figure 3.7.



(a) (b)

Figure 3.7: The numerical eigenvalues of the Sturm-Liouville problem (3.2) for the Calgary region. (b) Detailed view of the eigenvalues, showing that the eigenvalues are discrete.

From the results, it follows that the smallest eigenvalue is $\lambda_1 = -654499.7389238656$. However, Figure 3.7 also shows that we have both positive and negative eigenvalues, which is consistent with the Gershgorin circle theorem [15] (see Appendix C). As a consequence, it is possible that zero could be an eigenvalue, which implies that an orthogonality condition must be satisfied for the existence of an infinite number of solutions to the model. Since solutions do not exist unless the orthogonality condition is satisfied, and having an infinite number of solutions is not physically realistic, we examine whether zero is an eigenvalue. To do this we focus on the region near zero in Figure 3.8. This figure shows that zero is not an eigenvalue; the closest eigenvalues near zero are $\lambda_{807} = -828$ and $\lambda_{808} = 790$. By applying the Fredholm alternative, we therefore conclude that the solution to model (3.1) for the Calgary region exists and is unique for all forcing functions.

Theorem 3. *The solution to the boundary value problem in the Calgary*

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left((s-s^2)\frac{dV_0}{ds}\right) = k_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0, \end{cases}$$

exists and is unique for all forcing functions $k_0(s)$, provided that $\beta = \beta_{Calgary} \approx -654500$.

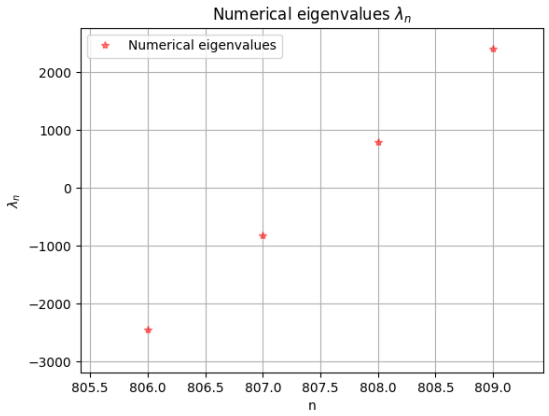


Figure 3.8: A closer look at the numerical eigenvalues of the Sturm-Liouville problem (3.2) for the Calgary region.

Similarly to the Gulf of Carpentaria, we compute the solution to model (3.1) for the Calgary region in the absence of a forcing term. The homogeneous solution is approximated using the finite difference method with $N = 2000$ points (see Appendix B.2), which gives the system

$$B\mathbf{y} = \mathbf{b}.$$

Just as in the Gulf of Carpentaria, the method is consistent and yields an invertible symmetric discretisation matrix (see Appendix D.2). However, unlike in the Gulf of Carpentaria, the method becomes unstable for Calgary. This is because the condition number $\kappa(B)$ grows as the stepsize h decreases (i.e. the number of points increases), as shown in Table 3.1. The condition number of a matrix is defined as [5] $\kappa(B) = ||B|| \cdot ||B^{-1}|| = \frac{|\lambda|_{\max}}{|\lambda|_{\min}}$, and a large value means that a small relative error in the right-hand side $\mathbf{b}$ may give a large relative error in the approximated solution $\mathbf{y}$ [15]. For $N = 2000$ points, the condition number of the discretisation matrix for Calgary is $\kappa_{\text{Calgary}} \approx 4232$, while for the Gulf of Carpentaria it is $\kappa_{\text{Carpentaria}}(B) \approx 2.25$. This large difference means that unlike in the Gulf of Carpentaria, the stepsize cannot be taken too small. As for very small stepsizes, the solutions become unreliable (chaotic) due to numerical instabilities. A possible explanation for this numerical instability is that the model is analytically unstable in Calgary, where $\beta < 0$.

Table 3.1: The condition number of the discretisation matrices for the Gulf of Carpentaria and Calgary for various number of points N.

Number of points (N)	Gulf of Carpentaria	Calgary
10	1.0000279521588649	1.0001363416096736
100	1.0031013708338066	1.0153577055745402
1000	1.3129698069058668	828.4806498161958
2000	2.252506081429955	4232.278481012858

The behaviour of sea breezes in the absence of a forcing term is plotted in Figure 3.9.

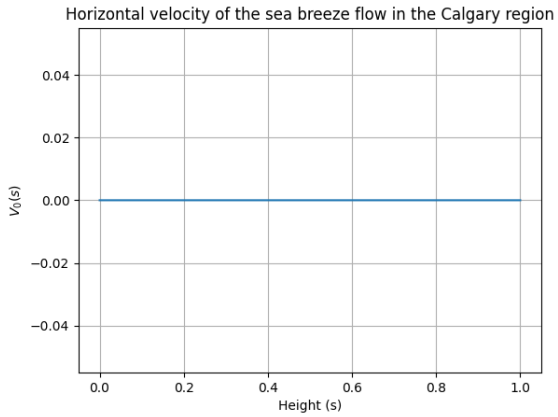


Figure 3.9: The horizontal velocity of the sea breeze in the Calgary region when there are no forcing terms.

From Figure 3.9, it is clear that just like in the Gulf of Carpentaria, sea breezes do not occur if there is no forcing term. So in the absence of external wind forces, sea breezes do not occur in the Calgary region. Suppose now that $\frac{K(s)}{\rho_0(s)} = 1$, which implies that the forcing function k_0 is constant and takes the form

$$k_0(s) = R_e\Big\{\cos^2(\alpha) + \frac{\sin^2(\alpha)}{C}\Big\}.$$

With the finite difference method with $N = 2000$ points, we compute the solution and the velocity profile is plotted in Figure 3.10.

[5] We use that the matrix is symmetric to express the condition number in terms of the eigenvalues.

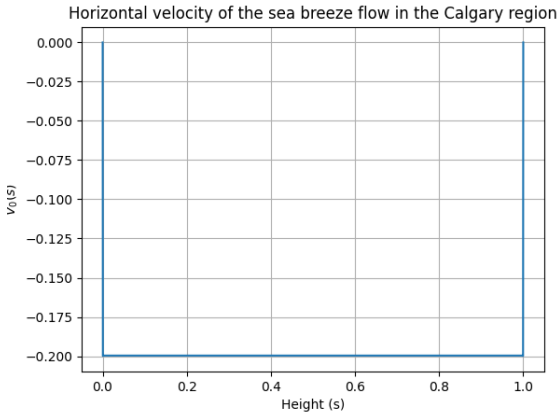


Figure 3.10: The horizontal velocity profile of the sea breeze in the Calgary region with a constant forcing function.

The velocity profile under the influence of a constant forcing function remains uniform over the interval $(0,1)$ as shown in Figure 3.12. However, unlike sea breezes in the Gulf of Carpentaria, the breezes in Calgary have negative velocity. These opposite flows result from the Coriolis force, which arises due to Earth's rotation. This force causes winds to be deflected to the right in the Northern Hemisphere and to the left in the Southern Hemisphere, relative to the direction of propagation [12], as illustrated in Figure 3.11. Since Calgary lies in the Northern Hemisphere and the Gulf of Carpentaria in the Southern Hemisphere, the Coriolis force acts in opposite directions in the two regions, causing the sea breezes to flow in opposite directions.
Furthermore, comparing Figure 3.5 with Figure 3.10, we see that the magnitude of the velocity in the Calgary region is greater than in the Gulf of Carpentaria. This difference in magnitude occurs because the Coriolis force is weakest near the equator increases with latitude [5]. Since the Gulf of Carpentaria lies closer to the equator than Calgary, the Coriolis force is weaker there, explaining the lower velocity.

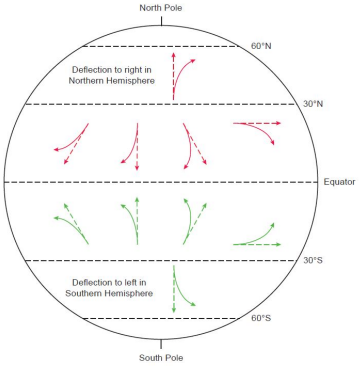


Figure 3.11: The deflection of winds in the Northern and Southern hemisphere caused by Coriolis forces (source: [12]).

We now consider model (3.1) for the same increasing quadratic forcing function given by Equation (3.15). Using the finite difference method with $N = 2000$ points (see Appendix B.2), the solution is plotted in Figure 3.12.

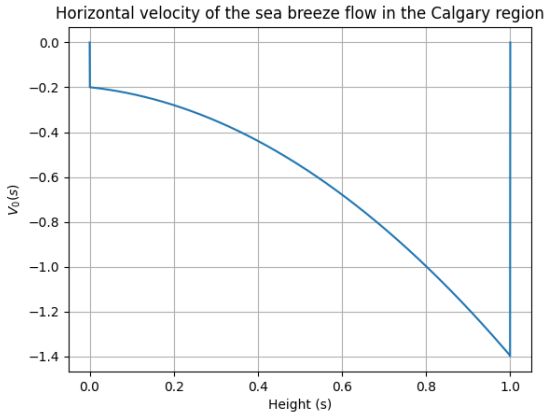


Figure 3.12: The horizontal velocity profile of the sea breeze in the Calgary region for a quadratic forcing function.

From Figure 3.12, we see that the velocity profile exhibits behaviour similar to that of the sea breeze in the Gulf of Carpentaria, where the horizontal velocity increases quadratically with height. The key differences are that breezes in Calgary flow in the opposite direction (i.e., it has negative velocity) and flow faster than the sea breezes in the Gulf of Carpentaria. These difference arise, because the Coriolis force acts in the opposite direction in Calgary compared to the Gulf of Carpentaria.

3.2. Physically relevant mass density function

The flow of air is not homogeneous and consists out of layers. So if we take a vertical slice of the flow and examine the mass flux, we find that the flux follows a cosine function. Thus, we assume that the physical mass density function has the following form:

$$\hat{m}(s) = \cos(\pi(2s-1)) + 1. \tag{3.16}$$

With this mass density function we write model (2.4) as:

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left((\cos(\pi(2s-1)) + 1)\frac{dV_0}{ds}\right) = k_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0. \end{cases} \tag{3.17}$$

The eigenvalue problem corresponding to model (3.17) is then given by:

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left((\cos(\pi(2s-1)) + 1)\frac{dV_0}{ds}\right) = \lambda V_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0. \end{cases} \tag{3.18}$$

This equation is rewritten as:

$$-\frac{d}{ds}\left((\cos(\pi(2s-1)) + 1)\frac{dV_0}{ds}\right) = \mu V_0(s), \tag{3.19}$$

where like in Section 3.1, the eigenvalues λ of the eigenvalue problem (3.18) satisfy

$$\lambda_n = \mu_n + \beta \text{ for } n \in \mathbb{N}.$$

In the same way as Section 3.1, we compute the eigenvalues with the finite difference method with $N = 2000$ points (see Appendix B.1). With the numerical eigenvalues, we study the solvability of the model (3.17) for the Gulf of Carpentaria and the Calgary region.

3.2.1. Gulf of Carpentaria

From [4], it follows that in the Gulf of Carpentaria, the parameter $\beta_{\text{Carpentaria}} = -\sigma S R_e$ is positive. The first 2000 eigenvalues of the eigenvalue problem (3.18) are computed using the finite difference method with $N = 2000$ points (see Appendix B.1), and the results are plotted in Figure 3.13. From the results it follows that the smallest eigenvalue is $\lambda_1 = 3192000.0040119556$, which agrees with the theoretical lower bound $\lambda_1 \geq$

$\beta_{\text{Carpentaria}}$ obtained via the Gershgorin circle theorem [15] (see Appendix C). Since zero is not an eigenvalue, we conclude by the Fredholm alternative [8] that the solution to model (3.17) exists and is unique for all forcing functions $k_0(s)$.

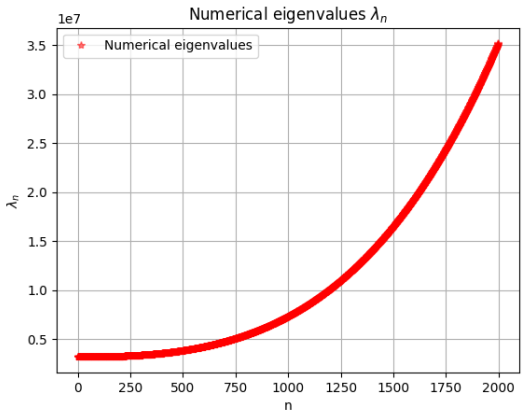


Figure 3.13: The numerical eigenvalues of Equation (3.18) for the Gulf of Carpentaria.

Theorem 4. *The solution to the boundary value problem in the Gulf of Carpentaria*

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\Big((\cos(\pi(2s-1))+1)\frac{dV_0}{ds}\Big) = k_0(s), \quad 0<s<1 \\ V_0(0) = V_0(1) = 0, \end{cases}$$

exists and is unique for all forcing functions $k_0(s)$, provided that $\beta = \beta_{Carpentaria} \approx 3192000$.

Proof. Suppose that $\lambda = 0$ is an eigenvalue, then the eigenvalue problem (3.18) is written as:

$$\beta V_0(s) - \frac{d}{ds}\Big(\cos(\pi(2s-1))+1)\frac{dV_0(s)}{ds}\Big) = 0.$$

Multiplying both sides by $V_0(s)$ and integrating gives:

$$\int_0^1 \beta V_0^2(s)\,ds - \int_0^1 \frac{d}{ds}\Big(\cos(\pi(2s-1))+1)\frac{dV_0(s)}{ds}\Big)V_0(s)\,ds = 0$$

$$\beta\int_0^1 V_0^2(s)\,ds - \big[\cos(\pi(2s-1))+1)V_0'(s)V_0(s)\big]_0^1 + \int_0^1 \cos(\pi(2s-1))+1)(V_0'(s))^2\,ds = 0.$$

Using the homogeneous Dirichlet boundary conditions we find:

$$\beta\int_0^1 V_0^2(s)\,ds - 0 + \int_0^1 \cos(\pi(2s-1)+1)(V_0'(s))^2\,ds = 0$$

$$\beta\int_0^1 V_0^2(s)\,ds = -\int_0^1 \cos(\pi(2s-1)+1)(V_0'(s))^2\,ds.$$

Since $\cos(\pi(2s-1)+1) \geq 0$ for $0 \leq s \leq 1$, and $(V_0'(s))^2 \geq 0$, we have that the right-hand side is non-positive, which implies that we have:

$$\beta\int_0^1 V_0^2(s)\,ds \leq 0. \tag{3.20}$$

Using Equation (3.20) and that $V_0^2(s) \geq 0$, we find that $0 \leq \int_0^1 V_0^2(s) \leq 0$. But this means that $\int_0^1 V_0^2(s) = 0$, which in turn means that $V_0(s) = 0$. Thus for $\lambda = 0$, we obtain the trivial solution, contradicting our assumption that 0 was an eigenvalue. By applying the Fredholm alternative [8], we conclude that the solution to Equation 3.18 exists and is unique for all forcing functions $k_0(s)$. □

Next, we study the solutions to model (3.17) with the mass density function $\hat{m}(s) = \cos(\pi(2s-1))+1$. The overall behaviour of this function is similar to that discussed in Section 3.1.1, but with one key difference. Near

the boundaries, this function changes more gradually compared to $s - s^2$, as illustrated in Figure 3.14. This occurs because the mass density function accounts for the fact that the factors causing the velocity to vanish at the boundaries, extend slightly into the interior of the domain. In other words, the boundary conditions exert a weak influence on the flow near the boundaries.

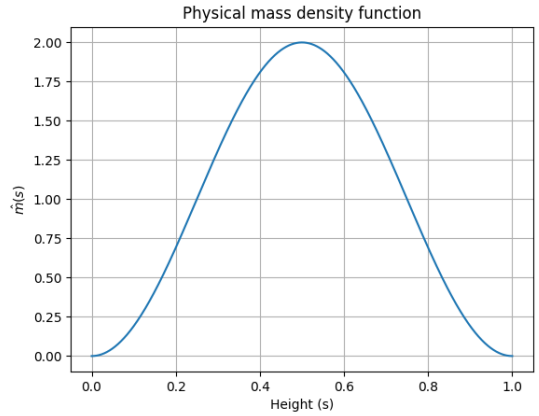


Figure 3.14: The physical mass density function used in model (3.17).

Similar to Section 3.1.1, we compute the solutions to model (3.17) for three different of the forcing function $k_0(s)$. We begin by computing the solution in the absence of a forcing function ($k_0(s) = 0$) using the finite difference method with $N = 2000$ points (see Appendix B.2), which gives the system $B\mathbf{y} = \mathbf{b}$. Similar to Section 3.1.1, the matrix B is invertible (this is proven by the python implementation provided in Appendix D.4) and $||B^{-1}||$ has an upper bound independent of the stepsize h. Combined with the local truncation error of order $\mathcal{O}(h^2)$ (see Appendix A), this implies that the method is convergent.
The solution to the model in the absence of wind forces, is plotted in Figure 3.15.

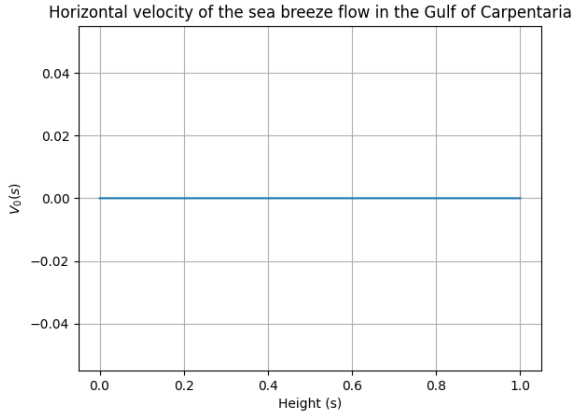


Figure 3.15: The horizontal velocity profile of the sea breeze in the Gulf of Carpentaria when no forcing terms are present.

From the figure, we see that without external wind forces, sea breezes do not occur in the Gulf of Carpentaria.

Next, we consider the solution for the constant forcing function given by Equation (3.14). The velocity profile is computed with the finite difference method (see Appendix B.2) with $N = 2000$ points and is plotted in Figure 3.17.

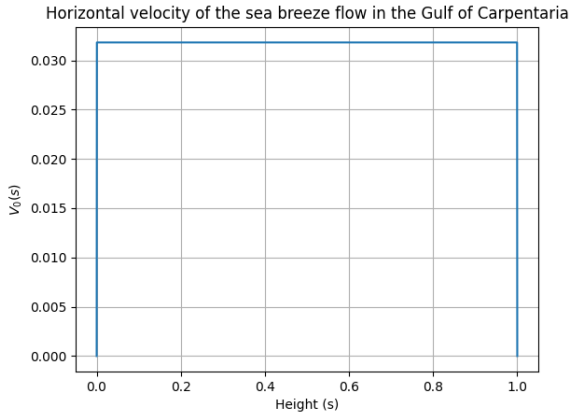


Figure 3.16: The horizontal velocity profile of the sea breeze in the Gulf of Carpentaria with a constant forcing function with the physical mass density function.

From the results, we see that like in Section 3.1.1, the horizontal velocity is uniform under the effect of constant wind forces. Since the mass of air will eventually move with the same speed, if we apply a constant force.

Lastly, we consider the quadratic forcing function given by Equation (3.15). With this forcing function, we compute the solution to the model with the finite difference method (see Appendix B.2). The resulting velocity profile is plotted in Figure 3.17.

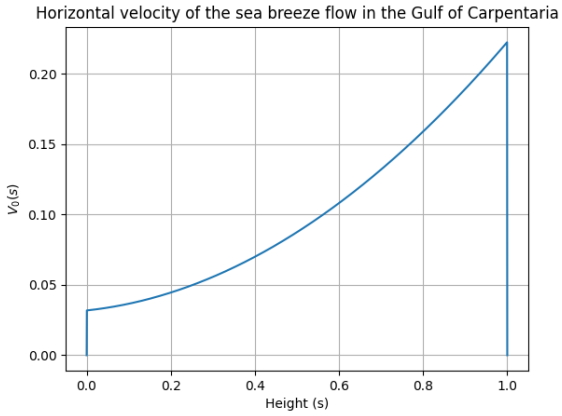
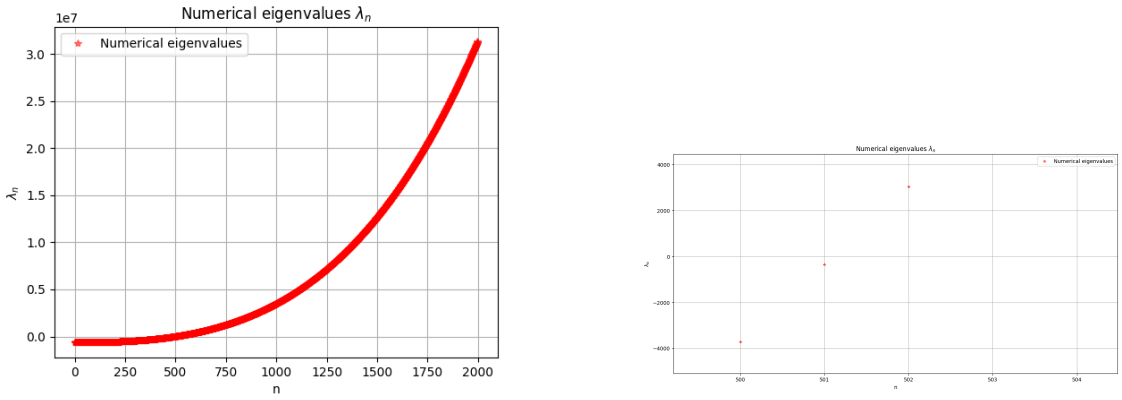


Figure 3.17: The horizontal velocity of the sea breeze in the Gulf of Carpentaria for an increasing quadratic forcing function with the physical density function.

The velocity profile under the quadratic forcing function grows quadratically with height. Similar to Section 3.1.1, we have that this behaviour occurs because the forcing increases with height, while the air density decreases. Consequently, the strongest forces act on the lightest air, resulting in the flow reaching its maximal velocity just below the thermal inversion layer.

3.2.2. Calgary region

From [4], it follows that the parameter β_{Calgary} is negative. In the same way as for the Gulf of Carpentaria, we compute the first 2000 eigenvalues of the eigenvalue problem (3.18) with $\beta < 0$. The numerical eigenvalues are then plotted in Figure 3.18.



(a) (b)

Figure 3.18: The numerical eigenvalues of Equation (3.18) for the Gulf of Carpentaria. (b) Detailed view of the eigenvalues, showing that the eigenvalues are discrete and that zero is not an eigenvalue.

From the results, it follows that the smallest eigenvalue is $\lambda_1 = -654499.9959880443$. However, similar to Section 3.1.2, we have both positive and negative eigenvalues, which is consistent with the Gershgorin circle theorem [15] (see Appendix C). For the same reasons as discussed in Section 3.1.2, we will determine whether zero is an eigenvalue. To determine this, we zoom in on the eigenvalues closest to zero, as illustrated in Figure 3.18b. This figure demonstrated that zero is not an eigenvalue; the eigenvalues closest to zero are $\lambda_{501} = -351.58482554$ and $\lambda_{502} = 3030.50962078$. The Fredholm alternative then implies that the solution to model (3.17) for the Calgary region exists and is unique for all forcing function k_0.

Theorem 5. *The solution to the boundary value problem in the Calgary region*

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left((\cos(\pi(2s-1))+1)\frac{dV_0}{ds}\right) = k_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0, \end{cases}$$

exists and is unique for all forcing functions $k_0(s)$, provided that $\beta = \beta_{Calgary} \approx -654500$.

In the same manner as in Section 3.2.1, we compute the solution to model (3.17) for the Calgary region using the three forcing functions introduced in Section 3.1. Figure 3.19 shows the solution in the absence of a forcing function, Figure 3.20 displays the velocity profile under constant forcing, and Figure 3.21 illustrates the resulting velocity under the quadratic forcing function. From these results, we observe that the solutions are identical to those presented in Section 3.1.2. In the first case we have that in the absence of wind forces, no sea breezes occur. In the second case, a constant forcing, leads to a constant velocity profile. Lastly, for quadratic forcing we have the the velocity increases with height, as the forcing grows with height while the air density decreases.

Similar to Section 3.1.2, the sea breezes in Calgary flow in the opposite direction compared to those in the Gulf of Carpentaria. This difference in the direction occurs because the Coriolis force acts in the opposite direction in the Northern Hemisphere and Southern Hemisphere [12]. Furthermore the breezes in Calgary are also faster. This is due to the fact that the Coriolis force is zero at the equator and increases with latitude [5].

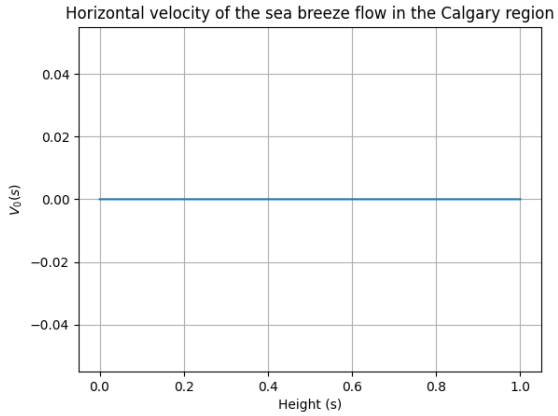


Figure 3.19: The horizontal velocity of the sea breeze in the Calgary region when there are no forcing terms.

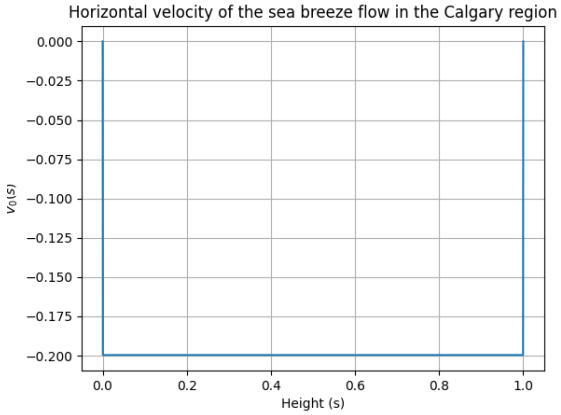


Figure 3.20: The horizontal velocity profile of the sea breeze in the Calgary region with a constant forcing function.

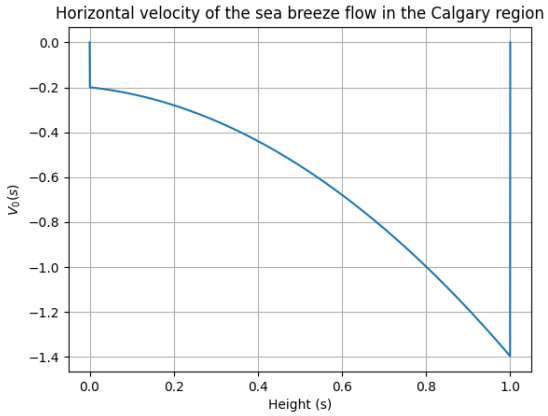


Figure 3.21: The horizontal velocity profile of the sea breeze in the Calgary region with a quadratic forcing function.

The solutions are computed using the finite difference method (see Appendix B.2) with $N = 2000$ points, which gives the system

$$B\mathbf{y} = \mathbf{b}.$$

From Appendix A we know that the central difference has a local truncation error of order $\mathcal{O}(h^2)$, which means that the method is consistent. But for the same reasons as for the Calgary region in Section 3.1.2, we have that the method is no longer stable. The condition number grows as the number of points used for the computation increases, as shown in Table 3.2. For $N = 2000$ points the condition number of the discretisation matrix for Calgary is $\kappa_{\text{Calgary}} \approx 89083$, while for the Gulf of Carpentaria it is $\kappa_{\text{Carpentaria}} \approx 11$. The large difference in the condition number means that unlike in the Gulf of Carpentaria, the stepsize for Calgary cannot be taken arbitrarily small. As for small stepsizes, the solutions become unreliable due to numerical instabilities. Similar to Section 3.1.2, we have that the numerical instability occurs, since the model itself is analytically unstable in Calgary.

Table 3.2: The condition number of the discretisation matrices for the Gulf of Carpentaria and Calgary for various number of points N.

Number of points (N)	Gulf of Carpentaria	Calgary
10	1.0002126035398937	1.0010379467168815
100	1.02467050290846	1.1367746747050993
1000	3.5023303694390795	9576.849406564716
2000	11.017190523492697	89083.40155822318

4

Conclusion and discussion

In this project, we studied a mathematical model describing the horizontal velocity of sea breeze flows for two physically relevant mass density functions. The behaviour of these functions is similar to the velocity profile of laminar flow in a closed pipe, where the velocity of the fluid is maximal in the centre and minimal near the walls. These functions were chosen to ensure that the solution to the model is unique, since multiple solutions would not be physically realistic. The analysis was carried out for two regions: the Gulf of Carpentaria (Australia) and the Calgary region (Canada). To confirm that the solutions to the model are indeed unique, we showed that zero is not an eigenvalue of the corresponding Sturm-Liouville problem. For the Gulf of Carpentaria, we proved this analytically, while for the Calgary region we demonstrated it numerically. Thus, the existence and uniqueness of the solutions follow immediately from the Fredholm alternative.

Using the mass density function $\hat{m}(s) = s - s^2$ and three different forcing functions, we applied the finite difference method to compute the velocity profiles of the breezes in the Gulf of Carpentaria. In the first case, with no forcing function, no sea breeze occurs because the wind forces responsible for generating sea breezes are absent. Secondly, under constant forcing, the velocity profile is constant, which makes physical sense since a constant force acting on the air will eventually result in the flow moving with the same speed everywhere. Lastly, for an increasing quadratic forcing function, the velocity increases quadratically with height and reaches its maximum just below the thermal inversion layer. This happens because the air density decreases with height, so the strongest forces act on the lightest air, while weaker forces push the heavier air near the surface.

In a similar manner, we computed the velocity profiles of the sea breezes in Calgary for the same three forcing functions. Although the overall behaviour is similar, the breezes in Calgary flow in the opposite direction. This difference occurs due to the Coriolis force, which acts in opposite directions in the Northern and Southern Hemispheres. Furthermore, the flows in Calgary are faster because the Coriolis force is zero at the equator and increases with latitude.

To better account for the effects of the boundaries on the flow, we also considered a second mass density function $\hat{m}(s) = \cos(\pi(2s-1)) + 1$. With the same three forcing functions, we then computed the solutions to the model. Surprisingly, the resulting velocity profiles were identical as those obtained with the previous mass density function, both in shape and magnitude. This outcome is expected, as both mass density functions exhibit similar parabolic behaviour and because the same forcing functions were applied in both cases.

To obtain more realistic results, future work could focus on deriving an explicit form of the forcing function $K(s)$ by studying the effects of global winds, such as the trade winds, westerlies, or Hadley cells on the formation of sea breezes. Furthermore, more realistic mass density functions could be obtained by determining realistic profiles for the air density and the viscosity in the two regions. Lastly, since the numerical method introduces approximation errors, we recommend proving analytically that zero is not an eigenvalue of the Sturm-Liouville problem corresponding to the model for the Calgary region (or any other region where the parameter β is negative).

A

Local truncation error in central difference

The central difference approximates the derivative of a function as

$$\frac{dy}{dx}|_i = \frac{y_{i+1} - y_{i-1}}{2h} + \mathcal{O}(h^2). \tag{A.1}$$

We will prove that the local truncation error of this method is of order $\mathcal{O}(h^2)$. Using the Taylor expansion of $y(x)$ about the point x_i we get:

$$\begin{aligned}
&y_{i+1} = y_i + hy_i' + \frac{h^2}{2!}y_i'' + \frac{h^3}{3!}y_i'' + \mathcal{O}(h^4)\\
&y_{i-1} = y_i - hy_i' + \frac{h^2}{2!}y_i'' - \frac{h^3}{3!}y_i'' + \mathcal{O}(h^4).\\
&\text{Then plugging these in Equation (A.1) gives us:}\\
&\frac{1}{2h}(y_i + hy_i' + \frac{h^2}{2!}y_i'' + \frac{h^3}{3!}y_i'' + \mathcal{O}(h^4) - (y_i - hy_i' + \frac{h^2}{2!}y_i'' - \frac{h^3}{3!}y_i'' + \mathcal{O}(h^4))) =\\
&\frac{1}{2h}(2hy_i' + \frac{h^3}{3}y_i'' + \mathcal{O}(h^4)) =\\
&y_i' + \mathcal{O}(h^2).
\end{aligned} \tag{A.2}$$

Proving that the central difference has a local truncation error of order $\mathcal{O}(h^2)$.
Thus in the case of the left hand-side of Equation (3.6) and writing $\hat{m}_i = s_i - s_i^2$, we get:

$$-\frac{d}{ds}\Big((s-s^2)\frac{dy}{ds}\Big)\Big|_i \approx -\frac{\hat{m}_{i+\frac{1}{2}}\frac{dy}{ds}|_{i+\frac{1}{2}} - \hat{m}_{i-\frac{1}{2}}\frac{dy}{ds}|_{i-\frac{1}{2}}}{h}. \tag{A.3}$$

We will show that the local truncation error of central difference method is order $\mathcal{O}(h^2)$ by using Taylor expansion of the functions. We expand the functions about the point s_i:

$$\begin{aligned}
\hat{m}_{i+\frac{1}{2}} &= \hat{m}_i + \frac{h}{2}\hat{m}_i' + \frac{h^2}{4}\hat{m}_i'' + \frac{h^3}{12}\hat{m}_i''' + \mathcal{O}(h^4)\\
\hat{m}_{i-\frac{1}{2}} &= \hat{m}_i - \frac{h}{2}\hat{m}_i' + \frac{h^2}{4}\hat{m}_i'' - \frac{h^3}{12}\hat{m}_i''' + \mathcal{O}(h^4)\\
y'_{i+\frac{1}{2}} &= y_i' + \frac{h}{2}y_i'' + \frac{h^2}{4}y_i''' + \frac{h^3}{12}y_i'''' + \mathcal{O}(h^4)\\
y'_{i-\frac{1}{2}} &= y_i' - \frac{h}{2}y_i'' + \frac{h^2}{4}y_i''' - \frac{h^3}{12}y_i'''' + \mathcal{O}(h^4).
\end{aligned} \tag{A.4}$$

Substituting these in the right hand-side of Equation (A.3) then gives:

$$
\begin{aligned}
&-\frac{1}{h}\Big\{(\hat{m}_i+\frac{h}{2}\hat{m}'_i+\frac{h^2}{4}\hat{m}''_i+\frac{h^3}{12}\hat{m}'''_i+\mathcal{O}(h^4))(y'_i+\frac{h}{2}y''_i+\frac{h^2}{4}y'''_i+\frac{h^3}{12}y''''_i+\mathcal{O}(h^4))-\\
&(\hat{m}_i-\frac{h}{2}\hat{m}'_i+\frac{h^2}{4}\hat{m}''_i-\frac{h^3}{12}\hat{m}'''_i+\mathcal{O}(h^4))(y'_i-\frac{h}{2}y''_i+\frac{h^2}{4}y'''_i-\frac{h^3}{12}y''''_i+\mathcal{O}(h^4))\Big\}.\\
&\text{This simplifies to:}\\
&-\frac{1}{h}\Big\{h\hat{m}_i y''_i+h\hat{m}'_i y'_i+\mathcal{O}(h^3)\Big\}=\\
&-\hat{m}_i y''_i-\hat{m}'_i y'_i+\mathcal{O}(h^2)=\\
&-\frac{d}{ds}(\hat{m}_i\frac{dy}{ds}\Big|_i)+\mathcal{O}(h^2).
\end{aligned}
\tag{A.5}
$$

The above proves that the central difference applied to Equation (A.3) has a local truncation error of order $\mathcal{O}(h^2)$.

B

Finite Difference Method

B.1. Eigenvalue problem

The eigenvalue problem corresponding to model (2.4) is given by:

$$\begin{cases} -\frac{d}{ds}\left(\hat{m}(s)\frac{dV_0}{ds}\right) = (\lambda-\beta)V_0(s) = \mu V_0(s), \quad 0<s<1 \\ V_0(0) = V_0(1) = 0. \end{cases} \tag{B.1}$$

For simplicity, we work with $\mu V_0(s)$ on the right hand-side for now. To compute the solution to this model, we apply the finite difference method. We begin by discretising the interval $[0,1]$ using N interior points as illustrated in Figure B.1.

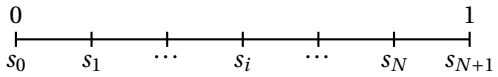


Figure B.1: Discretisation of the domain $(0,1)$ into discrete points s_i.

We define the grid points as $s_i = ih$, for $0 \le i \le N+1$ with stepsize $h = \frac{1}{N+1}$, such that $V_i = V(s_i)$, $\hat{m}_i = \hat{m}(s_i)$. For simplicity we will write $V_0(s) = V(s)$ to avoid any confusion with the indices. Let y_i be the numerical approximation of $V(s_i) = V_i$, then for $1 < i < N+1$ the discretised model can be written as:

$$\begin{cases} -\frac{d}{ds}\left(\hat{m}(s)\frac{dy}{ds}\right)\Big|_i = \mu y_i, \quad 0<s<1 \\ y_0 = y_{N+1} = 0. \end{cases} \tag{B.2}$$

To solve this equation we will apply central differences to **approximate** the derivates as shown in Equation (B.3), and for simplicity we use $\hat{m}_i$. Furthermore, we define $\hat{m}_{i+\frac{1}{2}} = \hat{m}(\frac{s_i+s_{i+1}}{2})$ and $\hat{m}_{i-\frac{1}{2}} = \hat{m}(\frac{s_i+s_{i-1}}{2})$.

$$\begin{aligned} &-\frac{\hat{m}_{i+\frac{1}{2}}\frac{dy}{ds}|_{i+\frac{1}{2}} - \hat{m}_{i-\frac{1}{2}}\frac{dy}{ds}|_{i-\frac{1}{2}}}{2\frac{h}{2}} = \mu y_i \\ &\text{Apply the central difference to the first derivatives.} \\ &\frac{-\hat{m}_{i+\frac{1}{2}}y_{i+1} + (\hat{m}_{i+\frac{1}{2}} + \hat{m}_{i-\frac{1}{2}})y_i - \hat{m}_{i-\frac{1}{2}}y_{i-1}}{h^2} = \mu y_i, \quad \text{for } 1<i<N. \end{aligned} \tag{B.3}$$

For $i = 1$ we obtain:

$$\begin{aligned} &\frac{-\hat{m}_{\frac{3}{2}}y_2 + (\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}})y_1 - \hat{m}_{\frac{1}{2}}y_0}{h^2} = \mu y_1 \\ &\text{Since } y_0 = 0 \text{ we find:} \\ &\frac{-\hat{m}_{\frac{3}{2}}y_2 + (\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}})y_1}{h^2} = \mu y_1. \end{aligned} \tag{B.4}$$

Similarly for $i = N$ we get:

$$\frac{(\hat{m}_{N+\frac{1}{2}} + \hat{m}_{N-\frac{1}{2}})y_N - \hat{m}_{N-\frac{1}{2}}y_{N-1}}{h^2} = \mu y_N. \tag{B.5}$$

Thus, we from the Finite Difference Method we obtain the system $A\mathbf{y} = \mu\mathbf{y}$, where A (an $N \times N$ matrix) and $\mathbf{y}$ are given by:

$$A = \begin{bmatrix} \frac{\hat{m}_{\frac{3}{2}}+\hat{m}_{\frac{1}{2}}}{h^2} & -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & 0 & 0 & 0 & 0 \\ -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & \frac{\hat{m}_{\frac{5}{2}}+\hat{m}_{\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & 0 & 0 & 0 \\ 0 & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & \frac{\hat{m}_{\frac{7}{2}}+\hat{m}_{\frac{5}{2}}}{h^2} & -\frac{\hat{m}_{\frac{7}{2}}}{h^2} & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{3}{2}}}{h^2} & \frac{\hat{m}_{N-\frac{1}{2}}+\hat{m}_{N-\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} \\ 0 & 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} & \frac{\hat{m}_{N+\frac{1}{2}}+\hat{m}_{N-\frac{1}{2}}}{h^2} \end{bmatrix}, \tag{B.6}$$

$$\mathbf{y}^T = \begin{bmatrix} y_1 & y_2 & \dots & y_N \end{bmatrix}^T. \tag{B.7}$$

By solving this eigenvalue problem and using that $\lambda = \mu + \beta$, we obtain the eigenvalues λ and eigenvectors of Equation (3.2). Furthermore, for the Gulf of Carpentaria we have $\beta_{\text{Carpentaria}} \approx 3.192 \cdot 10^6$ and for the Calgary region we have $\beta_{\text{Calgary}} \approx -6.545 \cdot 10^5$. It should be noted that the nth eigenvector represents the values of the nth eigenfunction at discrete points. The corresponding Python implementation for $\hat{m}(s) = s - s^2$ is provided in Appendix D.1 and the implementation for $\hat{m}(s) = \cos(\pi(2s-1)) + 1$ is given in Appendix D.3.

B.2. Boundary value problem

$$\begin{cases} \beta V_0(s) - \frac{d}{ds}\left(\hat{m}(s)\frac{dV_0}{ds}\right) = k_0(s), \quad 0 < s < 1 \\ V_0(0) = V_0(1) = 0. \end{cases} \tag{B.8}$$

To compute the solution to this model, we will use the Finite Difference method. To start we begin by discretising the interval $[0,1]$ by using N interior points, as seen in Figure B.1. We define the grid points as $s_i = ih$, for $0 \le i \le N+1$ with stepsize $h = \frac{1}{N+1}$, such that $V_i = V(s_i)$, $\hat{m}_i = \hat{m}(s_i)$ and $k_i = k(s_i)$.
For simplicity we will write $V_0(s) = V(s)$, $k_0(s) = k(s)$ to avoid any confusion with the indices. Let y_i be the numerical approximation of $V(s_i) = V_i$, then for $1 < i < N+1$ the discretised model can be written as:

$$\begin{cases} \beta y_i - \frac{d}{ds}\left(\hat{m}(s)\frac{dy}{ds}\right)\Big|_i = k_i, \quad 0 < s < 1 \\ y_0 = y_{N+1} = 0. \end{cases} \tag{B.9}$$

To solve this equation we will apply central differences to **approximate** the derivates as shown in Equation (B.10), and for simplicity we use $\hat{m}_i$. Furthermore, we define $\hat{m}_{i+\frac{1}{2}} = \hat{m}(\frac{s_i+s_{i+1}}{2})$ and $\hat{m}_{i-\frac{1}{2}} = \hat{m}(\frac{s_i+s_{i-1}}{2})$.

$$\begin{aligned} &\beta y_i - \frac{\hat{m}_{i+\frac{1}{2}}\frac{dy}{ds}|_{i+\frac{1}{2}} - \hat{m}_{i-\frac{1}{2}}\frac{dy}{ds}|_{i-\frac{1}{2}}}{2\frac{h}{2}} = k_i \\ &\text{Apply the central difference to the first derivatives.} \\ &\beta y_i + \frac{-\hat{m}_{i+\frac{1}{2}}y_{i+1} + (\hat{m}_{i+\frac{1}{2}} + \hat{m}_{i-\frac{1}{2}})y_i - \hat{m}_{i-\frac{1}{2}}y_{i-1}}{h^2} = k_i, \quad \text{for } 1 < i < N. \end{aligned} \tag{B.10}$$

For $i = 1$ we obtain:

$$\begin{aligned} &\beta y_1 + \frac{-\hat{m}_{\frac{3}{2}}y_2 + (\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}})y_1 - \hat{m}_{\frac{1}{2}}y_0}{h^2} = k_1 \\ &\text{Since } y_0 = 0 \text{ we find:} \\ &\beta y_1 + \frac{-\hat{m}_{\frac{3}{2}}y_2 + (\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}})y_1}{h^2} = k_1. \end{aligned} \tag{B.11}$$

Similarly for $i = N$ we get:

$$\beta y_N + \frac{(\hat{m}_{N+\frac{1}{2}} + \hat{m}_{N-\frac{1}{2}})\, y_N - \hat{m}_{N-\frac{1}{2}}\, y_{N-1}}{h^2} = k_N. \tag{B.12}$$

Thus, we obtain the system $B\mathbf{y} = \mathbf{b}$. Where A (an $N \times N$ matrix), $\mathbf{y}$ and $\mathbf{b}$ are given by:

$$B = \begin{bmatrix} \beta + \frac{\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}}}{h^2} & -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & 0 & 0 & 0 & 0 \\ -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & \beta + \frac{\hat{m}_{\frac{5}{2}} + \hat{m}_{\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & 0 & 0 & 0 \\ 0 & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & \beta + \frac{\hat{m}_{\frac{7}{2}} + \hat{m}_{\frac{5}{2}}}{h^2} & -\frac{\hat{m}_{\frac{7}{2}}}{h^2} & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{3}{2}}}{h^2} & \beta + \frac{\hat{m}_{N-\frac{1}{2}} + \hat{m}_{N-\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} \\ 0 & 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} & \beta + \frac{\hat{m}_{N+\frac{1}{2}} + \hat{m}_{N-\frac{1}{2}}}{h^2} \end{bmatrix}, \tag{B.13}$$

$$\mathbf{y}^T = \begin{bmatrix} y_1 & y_2 & \dots & y_N \end{bmatrix}^T, \quad \mathbf{b} = \begin{bmatrix} k_1 & k_2 & \dots & k_N \end{bmatrix}^T. \tag{B.14}$$

Solving this system yields the solution to the model equation. The corresponding Python implementation to numerically compute the solution for $\hat{m}(s) = s - s^2$ is provided in Appendix D.2. The Python implementation for the solution when $\hat{m}(s) = \cos(\pi(2s-1)) + 1)$ is provided in Appendix D.4.

C

Gershgorin circle theorem

After using the finite difference method to obtain the solution to model (2.4) we obtain the system $B\mathbf{y} = \mathbf{b}$, where the discretisation matrix B is given by (see Appendix B.2):

$$B = \begin{bmatrix} \beta + \frac{\hat{m}_{\frac{3}{2}} + \hat{m}_{\frac{1}{2}}}{h^2} & -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & 0 & 0 & 0 & 0 \\ -\frac{\hat{m}_{\frac{3}{2}}}{h^2} & \beta + \frac{\hat{m}_{\frac{5}{2}} + \hat{m}_{\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & 0 & 0 & 0 \\ 0 & -\frac{\hat{m}_{\frac{5}{2}}}{h^2} & \beta + \frac{\hat{m}_{\frac{7}{2}} + \hat{m}_{\frac{5}{2}}}{h^2} & -\frac{\hat{m}_{\frac{7}{2}}}{h^2} & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{3}{2}}}{h^2} & \beta + \frac{\hat{m}_{N-\frac{1}{2}} + \hat{m}_{N-\frac{3}{2}}}{h^2} & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} \\ 0 & 0 & 0 & 0 & -\frac{\hat{m}_{N-\frac{1}{2}}}{h^2} & \beta + \frac{\hat{m}_{N+\frac{1}{2}} + \hat{m}_{N-\frac{1}{2}}}{h^2} \end{bmatrix}. \tag{C.1}$$

Theorem 6 (Gershgorin circle theorem [15])**.** *The eigenvalues of a general $N \times N$ matrix A are located in the complex plane in the union of circles given by*

$$|z - a_{ii}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{ij}|, \ where \ z \in \mathbb{C}.$$

Note that the matrix is symmetric, which implies that the eigenvalues are real. Using the Gershgorin circle theorem, we find that the eigenvalues are in the interval given by

$$\begin{aligned} \beta + \frac{\hat{m}_{\frac{1}{2}}}{h^2} \leq \lambda \leq \beta + \frac{\hat{m}_{\frac{1}{2}}}{h^2} + \frac{2\hat{m}_{\frac{3}{2}}}{h^2} \\ \beta \leq \lambda \leq \beta + \frac{2\hat{m}_{j+\frac{1}{2}}}{h^2} + \frac{2\hat{m}_{j-\frac{1}{2}}}{h^2} \text{ for } 1 < j < N \\ \beta + \frac{\hat{m}_{N+\frac{1}{2}}}{h^2} \leq \lambda \leq \beta + \frac{\hat{m}_{N+\frac{1}{2}}}{h^2} + \frac{2\hat{m}_{N-\frac{1}{2}}}{h^2}. \end{aligned} \tag{C.2}$$

If β is positive, then it is clear that the Gershgorin circle theorem implies that all eigenvalues are positive. Furthermore, we have that the lower bound for the smallest eigenvalue is then given by $\lambda_{\min} \geq \beta$. Since the matrix B is symmetric this then implies that $||B^{-1}|| = \frac{1}{|\lambda|_{\min}} = \frac{1}{\lambda_{\min}} \leq \frac{1}{\beta}$. So we see that $||B^{-1}||$ has an upper bound independent of h, using [15] it follows that the method is stable.
On the other hand, if β is negative, then the Gershgorin circle theorem does not give information about the lower bound for the smallest absolute eigenvalue. Since β is negative, we have that interval in which we can find the eigenvalues, contains both positive and negative values. Thus, in the case that $\beta < 0$, the Gershgorin circle theorem does not provide an upper bound for $||B^{-1}||$, and it does not exclude the possibility that zero is an eigenvalue.

D

Implementation in Python

The code used that is used to compute the eigenvalues, the numerical solutions, and that the discretisation matrices are invertible can be found in this Appendix. It should be noted that actual comments are given by ###, while uncommented code is only given by a single #.

D.1. Eigenvalues and eigenfunction of the Hypergeometric equation

```
### Import area
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigh_tridiagonal  ### Function computes eigenvalues of tridiagonal matrix more efficiently


def FDM_Hypergeometric(h):
    """
    Description
    Function that computes the solution of the Hypergeometric equation (SL-Problem) using the Finite difference method,
    since this is a SL-Problem, this translates to finding the eigenvalues and eigenvectors of the problem.
    To compute the derivatives, the central difference approximation has been used, with error O(h^2).
    For the numerical implementation we did NOT write out the term of the form d/dx[f(x)g(x)], and for the discretization
    have left it in this form in order to preserve symmetry of the discretization matrix.

    The problem we are solving is L[y(x)] = mu * y(x)
    For our model we have instead L[y(x)] = (lambda - beta) y(x)
    We are interestsed in the eigenvalue lambda of OUR model, so lambda = mu + beta

    The domain of this SL-Problem is: 0 < x < 1,
    With homogeneuous Dirichlet Boundary conditions: V(0) = V(1) = 0.

    The result of the FDM gives Av = mu*v, where A is a SYMMETRIC tridiagonal matrix, which is of the form:
    [[M U X]
     [L M U]
     [X L M]],
    where M: Main diagonal, U: Upper diagonal, L: Lower diagonal, X: zeroes.

    For N interior points A is a N x N Matrix.
    NOTE: Since A is a symmetric tridiagonal matrix we can use the eigh_tridiagon function, which is much faster than the numpy linalg solver.
    NOTE: eigh_tridiagon(maindiagonal entries, offdiagonal entries), where since A is symmetric it does not matter if we use upper or lower diagonal
    NOTE: eigh_tridiagon returns the sorted eigenvalues (from smallest to highest) and the NORMALIZED eigenvectors, where the eigenvectors are the columns of the returned matrix
```

```
33
34      N: The number of interior points used in the discretization
35      h: Stepsize
36      p1(x) = x - x^2: The coefficient in front of the v_i's
37      eigenvalues: List of ordered eigenvalues from smallest to largest
38      eigenvectors: Matrix containing the normalized eigenvectors as columns, where
        column i corresponds to eigenvalue i
39      """
40      ### Create the empty diagonals of the discretization matrix,
41      ### where the upper and lower diagonals are one size smaller than the main diagonal
42      mainDiagonal = np.zeros(N)
43      upperDiagonal = np.zeros(N-1)
44      lowerDiagonal = np.zeros(N-1)
45
46      ### Next we fill in the values of the diagonal entries, for this we use a for loop
47      ### NOTE: x_i = i*h
48      ### NOTE: Python start counting from i=0, so when assigning the values to the
        diagonals we substract 1 from the index
49      for i in range(1, N+1):
50          ### Take the value of x at a half index as the average of the point before and
        after
51          x_for = ((h*i) + (h*(i+1))) / 2  # x_(i + 1/2) = (x_i + x_i+1) / 2
52          x_bac = ((h*i) + (h*(i-1))) / 2  # x_(i - 1/2) = (x_i + x_i-1) / 2
53
54          ### NOTE: When assiging values to the lower diagonals we need to subtract
        anothe -1 from the index,
55          ### so we have for the lowers i-2, since the entries of the lower only appear
        after equation i=2.
56          if i == 1:  # Equation with BC at x=0 (FIRST EQUATION)
57              mainDiagonal[i-1] = (p2(x_for) + p2(x_bac)) / h**2  # For y_1
58              upperDiagonal[i-1] = -p2(x_for) / h**2 # For y_2
59              ### No lower diagonal entry in row 1 Since y_0 = y(-1) = 0
60          if i == N:  ### Equation with BC at x = 1 (LAST EQUATION)
61              mainDiagonal[i-1] = (p2(x_for) + p2(x_bac)) / h**2 # For y_N
62              lowerDiagonal[i-2] = -p2(x_bac) / h**2  # For y_N-1
63              ### No upper diagonal entry in row N, since y_N+1 = y(1) = 0
64          else: ### Equations corresponding to interior points
65              mainDiagonal[i-1] = (p2(x_for) + p2(x_bac)) / h**2  # For y_i
66              upperDiagonal[i-1] = -p2(x_for) / h**2  # For y_i+1
67              lowerDiagonal[i-2] = -p2(x_bac) / h**2  # For y_i-1
68
69      ### From the discretization we obtain the system Av = mu*y, i.e. eigenvalue/vector
        problem
70      ### Thus the solutions of the EVP are the EW's and EV's
71      eigenvalues, eigenvectors = eigh_tridiagonal(mainDiagonal, upperDiagonal)  #
        Computes the EV and EW
72      return eigenvalues, eigenvectors
73
74
75  def p2(x):
76      """
77      Description
78      Coefficient which is present in the discretization using FDM:
79      [-p_i+1/2 * y_i+1 + (p_i+1/2 + p_i-1/2) * y_i - p_i-1/2 * y_i-1] / h^2 = mu * y_i
80      """
81      ### Coeff for the case that we set both parameters to 0
82      return x - x**2
83
84
85  def plot_EF_Hypergeometric(eigval, eigvec, j, region):
86      """
87      Description
88      Plots the eigenfunction corresponding to the j-th eigenvalue
89      eigval: List of ordered eigenvalues
90      eigvec: Matrix containing the normalized eigenvectors as columns in the same order
        as the eigenvalues
91      j: Index of eigenvalues and eigenvectors
92
93      NOTE: The boundary values have NOT yet been added to the eigenfunction values!
94      """
95      eigenfunction = eigvec[:, j]  ### J-th column is jth eigenvector
```

```
96      eigenfunction = np.concatenate(([0], eigenfunction, [0]))  ### Add the BC to the
        start and end of the vector
97      x_as = np.linspace(0, 1, N+2)
98
99      plt.title("Eigenfunction for $\\lambda$" + str(j) + "=" + str(eigval[j]) + "\n"
100     + "(" + region + ")")
101     plt.plot(x_as, eigenfunction)
102     plt.xlabel("x")
103     plt.ylabel("V_0(x)")
104     plt.grid()
105     plt.show()
106
107
108     ### NOTE: Uncomment code below to plot multiple eigenfunctions in a single plot
109     eigenfunctions = []
110     for index in range(N):
111         eigenfunctions.append(np.concatenate(([0], eigvec[:, index], [0])))
112     plt.title("First five (numerical) eigenfunctions $\\phi_n(x)$")
113     for i in range(10):
114         plt.plot(x_as, eigenfunctions[i], label=fr'$\phi_{i}(x)$')
115     plt.xlabel("x")
116     plt.ylabel("$\\phi_n(x)$")
117     plt.legend()
118     plt.grid()
119     plt.show()
120
121
122 def plot_EW_Hypergeometric(eigval, region):
123     index_axis = range(N)
124     plt.title("Numerical eigenvalues $\\lambda_n$" "\n" + "(" + region + ")")
125     plt.plot(index_axis, eigval, '*', alpha = 0.5, label = "Numerical eigenvalues",
        color='red', zorder = 3)
126     plt.xlabel("n")
127     plt.ylabel("$\\lambda_n$")
128     plt.grid()
129     plt.legend()
130     plt.show()
131
132
133 def close_to_zero(eigval):
134     closest_indices_to_zero = np.argsort(np.abs(eigval))[:2]
135     closest_elements = eigval[closest_indices_to_zero]
136     print(closest_elements)
137
138
139
140 ### Work space
141 ### Default number of points is 2000
142 h = 1/2000
143 N = int((1/h) - 1)
144 print("Number of interior points =", N)
145
146 EW, EV = FDM_Hypergeometric(h)  # Computes the eigenvalues and eigenvectors (
        eigenfunctions)
147
148 ### Check if the vectors span whole of R_n, so they form a basis for the solution space
149 ### Uses that the EF are already independent and checks if They span the solution space
        ,
150 ### so in other words check if EF form a basis
151 ### NOTE: Should be faster since we don't need to compute the DET of large matrix
152 ### NOTE: This property of teh EF is not used in the report
153 rank = np.linalg.matrix_rank(EV)
154 if rank == EV.shape[1]:  # Rank of matrix == number of pivots
155     print("The eigenfunctions form a basis for the solution space.")
156 else:
157     print("The eigenfunctions do not form a basis.")
158
159
160 ### Check orthogonality of eigenfunction
161 # EF1 = EV[:, 421]
162 # EF2 = EV[:, 421]
```

```
# print(np.dot(EF1, EF2))

### Beta values = -sigma * S * R_e
beta_GulfCar = -133 * -0.24 * 10**5 # Value of beta in the Gulf of Carpentaria, beta > 0
beta_Calgary = -8.5 * 0.77 * 10**5 # Value of beta in the Calgary region, beta < 0

### Plot the EW and EF of the EVP, NOT THE EW OF THE MODEL
# print(min(EW))
# plot_EF_Hypergeometric(EW, EV, 0, "No region")
# plot_EW_Hypergeometric(EW, "No region")


### Compute the eigenvalues for the EVP of model in the project, where lambda = mu + beta
### Here mu is the eigenvalue of the SL-problem
EW_model_GulfCar = EW + beta_GulfCar
EW_model_Calgary = EW + beta_Calgary


### Plot the eigenvalues and eigenfunction for the Gulf of Carpentaria
### NOTE: The eigenfunction are the still the same, only the EW are different
print("Smallest eigenvalues =", min(EW_model_GulfCar))
plot_EF_Hypergeometric(EW_model_GulfCar, EV, 0, "Gulf of Carpentaria")
plot_EW_Hypergeometric(EW_model_GulfCar, "Gulf of Carpentaria")


### Plot the eigenvalues and eigenfunction for the Calgary region
### NOTE: The eigenfunction are the still the same, only the EW are different
print("Smallest eigenvalues =", min(EW_model_Calgary))
plot_EF_Hypergeometric(EW_model_Calgary, EV, 0, "Calgary Region")
plot_EW_Hypergeometric(EW_model_Calgary, "Calgary Region")
close_to_zero(EW_model_Calgary)
```

D.2. Solutions to the Hypergeometric model

```
# Import area
import numpy as np
import matplotlib.pylab as plt
from scipy.sparse.linalg import spsolve
from scipy.linalg import solve_banded
import scipy

"""
In this program we compute the solutions of the the model in Hypergeometric form, which is an ODE given by
b*y - [m_hat * y']' = 0, by using the Finite difference method.
Thus we are solving a BVP with y(0) = y(1) = 0.

We do this by two different approaches.
1. We do not work out the derivative
2. We fully work out the derivative, such that the Equation also contains a first order derivative

NOTE: m_hat is chosen such that the model EVP was transformed into the Hypergeometric equation,
with parameters alpha, gamma = 0.
NOTE: Actual comments are given by ###, the commented code is given by a single #
"""


def FDM_Model_Hypergeometric_1(h, b, region):
    """
    Description
    NOTE: Version 1 which computes the solution by NOT WORKING OUT THE DERIVATIVES, into the form
          y'' + y' + y = D and solving it in the form as shown.
          We do this to make sure the discretisation matrix A will be symmetric.

```

```
    Function computes the solution to the model equation (BVP), using the Finite
    difference method.
    We approximate the derivatives with the Central Differences.

    The domain of this SL-Problem is: 0 < x < 1,
    With homogeneuous Dirichlet Boundary conditions: V(0) = V(1) = 0.

    The result of the FDM gives Av = RHS, where A is a SYMMETRIC TRIDIAGONAL matrix,
    which is of the form:
    [[M U X]
     [L M U]
     [X L M]],
    where M: Main diagonal, U: Upper diagonal, L: Lower diagonal, X: zeroes.
    From FDM we have that the matrix only contains three nonzero diagonals

    For N interior points we have that A is a N x N matrix.

    For equation i we get that the discretisation gives:
    b*y_i + (1/h^2) * (-m_(i+1/2) * y_(i+1) + (m_(i+1/2) + m_(i-1/2)) * y_i - m_(i-1/2)
     * y_(i-1)) = RHS_i
    The coefficients in front of the y_j are put in the diagonals of the discretisation
     matrix.
    After FDM we obtain the system Ay = d, we then solve this for the vector y, where d
     is the zero vector

    Returns a vector which is the solution of Ay = d
    This vector represents the function values of the solution to the ODE
    NOTE: This vector does not contain the values at x=0 and x=1 YET! so we need to add
     them!!!

    N: Number of interior points
    h: Stepsize
    m_hat_hypergeometric: Mass density function for this model
    b: Beta value
    d: RHS of the system Ay = d
    y: Vector containing the function values of the solution at the internal gridpoints
    k0: The "normalized" forcing term
    """
    ### Create the empty diagonals of the discretization matrix,
    ### where the upper and lower diagonals are one size smaller than the main diagonal
    mainDiagonal = np.zeros(N)
    upperDiagonal = np.zeros(N-1)
    lowerDiagonal = np.zeros(N-1)

    ### Forcing Term
    d = np.zeros(N)  # RHS of the sytem Ay = d after applying FDM

    ### Next we compute the diagonal entries and plug them in the diagonals
    ### NOTE: x_i = i*h, where h = 1 / (N+1)
    ### NOTE: Python start counting from i=0, so when assigning the values to the
    diagonals we substract 1 from the index
    for i in range(1, N+1):
        ### Take the value of x at a half index as the average of the point before and
    after
        x_for = ((h*i) + (h*(i+1))) / 2  # x_(i + 1/2) = (x_i + x_i+1) / 2
        x_bac = ((h*i) + (h*(i-1))) / 2  # x_(i - 1/2) = (x_i + x_i-1) / 2

        ### NOTE: When assiging values to the lower diagonals we need to subtract
    anothe -1 from the index,
        ### so we have for the lowers i-2, since the entries of the lower only appear
    after equation i=2.
        if i == 1:  # Equation with BC at x=0 (FIRST EQUATION)
            mainDiagonal[i-1] = b + (m_hat_Hypergeometric(x_for) + m_hat_Hypergeometric
    (x_bac)) / h**2  # Coeff for y_1
            upperDiagonal[i-1] = -1 * m_hat_Hypergeometric(x_for) / h**2  # Coeff for
    y_2
            ### No lower diagonal entry in row 1 Since y_0 = y(0) = 0
        if i == N:  # Equation with BC at x = 1 (LAST EQUATION)
            mainDiagonal[i-1] = b + (m_hat_Hypergeometric(x_for) + m_hat_Hypergeometric
    (x_bac)) / h**2  # Coeff for y_N
```

```
            lowerDiagonal[i-2] = -1 * m_hat_Hypergeometric(x_bac) / h**2  # Coeff for y_N-1
            ### No upper diagonal entry in row N, since y_N+1 = y(1) = 0
        else: # Equations corresponding to interior points, so all 1 < i < N
            mainDiagonal[i-1] = b + (m_hat_Hypergeometric(x_for) + m_hat_Hypergeometric(x_bac)) / h**2  # Coeff for y_i
            upperDiagonal[i-1] = -1 * m_hat_Hypergeometric(x_for) / h**2  # Coeff for y_i+1
            lowerDiagonal[i-2] = -1 * m_hat_Hypergeometric(x_bac) / h**2  # Coeff for y_i-1

        d[i-1] = k0(i*h, region)  # Assigns the values of the forcing term k0 to the RHS vector

    ### Puts the values in a vector and turns them into the diagonals of the matrx
    mainDiagonalMatrix = np.diag(mainDiagonal)
    upperDiagonalMatrix = np.diag(upperDiagonal, 1)
    lowerDiagonalMatrix = np.diag(lowerDiagonal, -1)
    A = mainDiagonalMatrix + upperDiagonalMatrix + lowerDiagonalMatrix  # Discretisation matrix

    ### Compute the condition number of the matrix (TAKES LONG)
    ### NOTE: Important for symmetric matrices we take the maximal absolute value!!! so |L|_max and not |L_max|!!!
    eigenwaarden = np.linalg.eig(A)[0]
    conditionNumber2 = np.linalg.cond(A)
    print("Condition number (other method) =", conditionNumber2)

    ### Check symmetry and Stability of the method
    ### NOTE: The norm of the inverse increases as h->0, so unsure if there exist some constant that is
    ### always bigger than the norm of the inverse matrix.
    print("Matrix is symmetric =", scipy.linalg.issymmetric(A))
    NormInverse_A = 1 / np.min(np.abs(eigenwaarden))  # Since A is symmetric
    print("Smallest absolute EW =", np.min(np.abs(eigenwaarden)))
    print("||A^-1|| =", NormInverse_A)

    ### Check if matrix is invertible, by checking if columns of matrix span R^n
    ### We do this by checking the rank of the matrix
    rank = np.linalg.matrix_rank(A)
    if rank == A.shape[0]:  # If number of pivots is equal to N of the NxN matrix, columns are independent
        print("The columns of the matrix span R^n, hence the matrix is invertible")
    else:
        print("Columns do not span R")


    ### Matrix A is symmetric and tridiagonal so we can use more efficient function to solve Ay = d
    # y = np.linalg.solve(A, d)  # Slower method
    # y = spsolve(A, d)  # Faster method

    ### Band method (fastest method)
    diagonals = np.zeros((3, N))
    diagonals[0, 1:] = upperDiagonal
    diagonals[1, :] = mainDiagonal
    diagonals[2, :-1] = lowerDiagonal
    y = solve_banded((1,1), diagonals, d)

    # print("A =", A)
    # print("y = ", y)

    ### Add the two boundary values at x=1 and x=0 to the solution values
    y = np.concatenate(([0], y, [0]))
    return y


def m_hat_Hypergeometric(x):
    """
    Description
```

```
    The mass density function that was used to transform the model EVP into the
    Hypergeometric Equation.
    """
    return (x - x**2)


def k0(x, region):
    """
    Description
    The forcing term which is on the RHS of the simplified model (written in terms of s
    )
    These are just some trivial guesses!
    """

    if region == "Carpentaria":
        ### Forcing Term (Gulf of Carpentaria)
        # res = R_e * ((np.cos(alpha_Gulfcar))**2 + (np.sin(alpha_Gulfcar))**2 /
    C_Gulfcar) * 0  # No forcing
        # res = R_e * ((np.cos(alpha_Gulfcar))**2 + (np.sin(alpha_Gulfcar))**2 /
    C_Gulfcar) * 1  # Constant forcing
        res = R_e * ((np.cos(alpha_Gulfcar))**2 + (np.sin(alpha_Gulfcar))**2 /
    C_Gulfcar) * (5*x**2 + x + 1)  # Quadratic forcing
    elif region == "Calgary":
        ### Forcing Term (Gulf of Carpentaria)
        # res = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 /
    C_Calgary) * 0  # No forcing
        # res = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 /
    C_Calgary) * 1  # Constant forcing
        res = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 /
    C_Calgary) * (5*x**2 + x + 1)  # Quadratic forcing
    return res


### Work space
"""
Default stepsize we have used is 1/2000
"""
h = 1/2000  # Stepsize
N = int((1/h) - 1)  # Number of interior points
R_e = 10**5  # Reynolds number for the model
print("Number of interior points =", N)

### Beta values = -sigma * S * R_e
beta_GulfCar = -133 * -0.24 * 10**5 # Value of beta in the Gulf of Carpentaria, beta >
    0
beta_Calgary = -8.5 * 0.77 * 10**5 # Value of beta in the Calgary region, beta < 0
# print(beta_GulfCar)
# print(beta_Calgary)


### Parameter values
sigma_Gulfcar = 133
C_Gulfcar = 0.97
S_Gulfcar = -0.24
alpha_Gulfcar = np.pi * 5 / 4  # Breeze propagating in SW-Direction

sigma_Calgary = 8.5
C_Calgary = 0.62
S_Calgary = 0.77
alpha_Calgary = np.pi / 4  # Breeze propagating in NW-Direction

### Computes the solutions with different variations of FDM
### NOTE: Uncomment the line for which you want to compute the solution for the region
sol = FDM_Model_Hypergeometric_1(h, beta_GulfCar, "Carpentaria")  # Solves ODE in the
    original form as the model
# sol = FDM_Model_Hypergeometric_1(h, beta_Calgary, "Calgary")

x_as = np.linspace(0,1,N+2)
plt.plot(x_as, sol)
plt.title("Horizontal velocity of the sea breeze flow in the Gulf of Carpentaria")
# plt.title("Horizontal velocity of the sea breeze flow in the Calgary region")
```

```
plt.xlabel("Height (s)")
plt.ylabel(r'$V_0(s)$')
plt.grid()
plt.show()

### Plot the mass density function
plt.plot(x_as, m_hat_Hypergeometric(x_as))
plt.title("Mass density function for the hypergeometric model")
plt.xlabel("Height (s)")
plt.ylabel(r'$\hat{m}(s)$')
plt.grid()
plt.show()
```

D.3. Eigenvalues of the model with a physical mass density function

```
# Import area
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigh_tridiagonal  # Function computes eigenvalues of
    tridiagonal matrix more efficiently


def FDM_Physical(h):
    """
    Description
    Function that computes the solution of the Equation with a physical mass density
    funtion (SL-Problem) using the Finite difference method,
    since this is a SL-Problem, this translates to finding the eigenvalues and
    eigenvectors of the problem.
    To compute the derivatives, the central difference approximation has been used,
    with error O(h^2).
    For the numerical implementation we did NOT write out the term of the form d/dx[f(x
    )g(x)], and for the discretization
    have left it in this form in order to preserve symmetry of the discretization
    matrix.

    The problem we are solving is L[y(x)] = mu * y(x)
    For our model we have instead L[y(x)] = (lambda - beta) y(x)
    We are interestsed in the eigenvalue lambda of OUR model, so lambda = mu + beta

    The domain of this SL-Problem is: 0 < x < 1,
    With homogeneuous Dirichlet Boundary conditions: V(0) = V(1) = 0.

    The result of the FDM gives Av = mu*v, where A is a SYMMETRIC tridiagonal matrix,
    which is of the form:
    [[M U X]
     [L M U]
     [X L M]],
    where M: Main diagonal, U: Upper diagonal, L: Lower diagonal, X: zeroes.

    For N interior points A is a N x N Matrix.
    NOTE: Since A is a symmetric tridiagonal matrix we can use the eigh_tridiagon
    function, which is much faster than the numpy linalg solver.
    NOTE: eigh_tridiagon(maindiagonal entries, offdiagonal entries), where since A is
    symmetric it does not matter if we use upper or lower diagonal
    NOTE: eigh_tridiagon returns the sorted eigenvalues (from smallest to highest) and
    the NORMALIZED eigenvectors, where the eigenvectors are the columns of the returned
     matrix

    N: The number of interior points used in the discretization
    h: Stepsize
    m_hat_physical = cos(x): The coefficient in front of the v_i's
    eigenvalues: List of ordered eigenvalues from smallest to largest
    eigenvectors: Matrix containing the normalized eigenvectors as columns, where
    column i corresponds to eigenvalue i
    """
    ### Create the empty diagonals of the discretization matrix,
    ### where the upper and lower diagonals are one size smaller than the main diagonal
    mainDiagonal = np.zeros(N)
    upperDiagonal = np.zeros(N-1)
```

```
    lowerDiagonal = np.zeros(N-1)

    ### Next we fill in the values of the diagonal entries, for this we use a for loop
    ### NOTE: x_i = i*h
    ### NOTE: Python start counting from i=0, so when assigning the values to the diagonals we substract 1 from the index
    for i in range(1, N+1):
        ### Take the value of x at a half index as the average of the point before and after
        x_for = ((h*i) + (h*(i+1))) / 2  # x_(i + 1/2) = (x_i + x_i+1) / 2
        x_bac = ((h*i) + (h*(i-1))) / 2  # x_(i - 1/2) = (x_i + x_i-1) / 2

        ### NOTE: When assiging values to the lower diagonals we need to subtract anothe -1 from the index,
        ### so we have for the lowers i-2, since the entries of the lower only appear after equation i=2.
        if i == 1:  # Equation with BC at x=0 (FIRST EQUATION)
            mainDiagonal[i-1] = (m_hat_physical(x_for) + m_hat_physical(x_bac)) / h**2  # For y_1
            upperDiagonal[i-1] = -m_hat_physical(x_for) / h**2 # For y_2
            ### No lower diagonal entry in row 1 Since y_0 = y(-1) = 0
        if i == N:  # Equation with BC at x = 1 (LAST EQUATION)
            mainDiagonal[i-1] = (m_hat_physical(x_for) + m_hat_physical(x_bac)) / h**2  # For y_N
            lowerDiagonal[i-2] = -m_hat_physical(x_bac) / h**2  # For y_N-1
            # No upper diagonal entry in row N, since y_N+1 = y(1) = 0
        else: # Equations corresponding to interior points
            mainDiagonal[i-1] = (m_hat_physical(x_for) + m_hat_physical(x_bac)) / h**2  # For y_i
            upperDiagonal[i-1] = -m_hat_physical(x_for) / h**2  # For y_i+1
            lowerDiagonal[i-2] = -m_hat_physical(x_bac) / h**2  # For y_i-1

    ### Method 1 for computing EW and EV, where at the end we sort from smallest to largest
    ### NOTE: Use this if Method 2 is not convergin
    # mainDiagonalMatrix = np.diag(mainDiagonal)
    # upperDiagonalMatrix = np.diag(upperDiagonal, 1)
    # lowerDiagonalMatrix = np.diag(lowerDiagonal, -1)
    # A = mainDiagonalMatrix + upperDiagonalMatrix + lowerDiagonalMatrix
    # eigenvalues, eigenvectors = np.linalg.eig(A)
    # idx = np.argsort(eigenvalues.real)
    # eigenvalues = eigenvalues[idx]
    # eigenvectors = eigenvectors[:, idx]

    ### Method 2 for computing the EW and EV
    ### From the discretization we obtain the system Av = mu*y, i.e. eigenvalue/vector problem
    ### Thus the solutions of the EVP are the EW's and EV's
    eigenvalues, eigenvectors = eigh_tridiagonal(mainDiagonal, upperDiagonal)  # Computes the EV and EW
    return eigenvalues, eigenvectors


def m_hat_physical(x):
    """
    Description
    Coefficient which is present in the discretization using FDM:
    [-p_i+1/2 * y_i+1 + (p_i+1/2 + p_i-1/2) * y_i - p_i-1/2 * y_i-1] / h^2 = mu * y_i
    NOTE: Here we use a physical mass density function of form cosine
    """
    y = np.cos(np.pi * (2*x-1)) + 1
    return y


def plot_EF_Hypergeometric(eigval, eigvec, j, region):
    """
    Description
    Plots the eigenfunction corresponding to the j-th eigenvalue
    eigval: List of ordered eigenvalues
    eigvec: Matrix containing the normalized eigenvectors as columns in the same order as the eigenvalues
```

```
    j: Index of eigenvalues and eigenvectors

    NOTE: The boundary values have NOT yet been added to the eigenfunction values!
    """
    eigenfunction = eigvec[:, j]  # J-th column is jth eigenvector
    eigenfunction = np.concatenate(([0], eigenfunction, [0]))  # Add the BC to the start and end of the vector
    x_as = np.linspace(0, 1, N+2)

    plt.title("Eigenfunction for $\\lambda$" + str(j) + "=" + str(eigval[j]) + "\n" + "(" + region + ")")
    plt.plot(x_as, eigenfunction)
    plt.xlabel("x")
    plt.ylabel("V_0(x)")
    plt.grid()
    plt.show()


    ### NOTE: Uncomment code below to plot multiple eigenfunctions in a single plot
    eigenfunctions = []
    for index in range(N):
        eigenfunctions.append(np.concatenate(([0], eigvec[:, index], [0])))
    plt.title("First five (numerical) eigenfunctions $\\phi_n(x)$")
    for i in range(5):
        plt.plot(x_as, eigenfunctions[i], label=fr'$\phi_{i}(x)$')
    plt.xlabel("x")
    plt.ylabel("$\\phi_n(x)$")
    plt.legend()
    plt.grid()
    plt.show()


def plot_EW_Hypergeometric(eigval, region):
    index_axis = range(N)
    plt.title("Numerical eigenvalues $\\lambda_n$" + "\n" + "(" + region + ")")
    plt.plot(index_axis, eigval, '*', alpha = 0.5, label = "Numerical eigenvalues", color='red', zorder = 3)
    plt.xlabel("n")
    plt.ylabel("$\\lambda_n$")
    plt.grid()
    plt.legend()
    plt.show()


def close_to_zero(eigval):
    closest_indices_to_zero = np.argsort(np.abs(eigval))[:2]
    closest_elements = eigval[closest_indices_to_zero]
    print(closest_elements)



### Work space
### Default number of points is 2000
h = 1/2000
N = int((1/h) - 1)
print("Number of interior points =", N)

EW, EV = FDM_Physical(h)  # Computes the eigenvalues and eigenvectors (eigenfunctions)


### Check if the vectors span whole of R_n, so they form a basis for the solution space
### Uses that the EF are already independent and checks if They span the solution space,
### NOTE: Should be faster since we don't need to compute the DET of large matrix
### NOTE: This property of teh EF is not used in the report
rank = np.linalg.matrix_rank(EV)
if rank == EV.shape[1]:  # Rank of matrix == number of pivots
    print("The eigenfunctions form a basis for the solution space.")
else:
    print("The eigenfunctions do not form a basis.")

```

```
### Check orthogonality of eigenfunction
# EF1 = EV[:, 421]
# EF2 = EV[:, 421]
# print(np.dot(EF1, EF2))

### Beta values = -sigma * S * R_e
beta_GulfCar = -133 * -0.24 * 10**5 # Value of beta in the Gulf of Carpentaria, beta > 0
beta_Calgary = -8.5 * 0.77 * 10**5 # Value of beta in the Calgary region, beta < 0

### Plot the EW and EF of the EVP, NOT THE EW OF THE MODEL
# print(min(EW))
# plot_EF_Hypergeometric(EW, EV, 0, "No region")
# plot_EW_Hypergeometric(EW, "No region")


### Compute the eigenvalues for the EVP of model in the project, where lambda = mu + beta
### Here mu is the eigenvalue of the SL-problem
EW_model_GulfCar = EW + beta_GulfCar
EW_model_Calgary = EW + beta_Calgary


### Plot the eigenvalues and eigenfunction for the Gulf of Carpentaria
### NOTE: The eigenfunction are the still the same, only the EW are different
print("Smallest eigenvalues =", min(EW_model_GulfCar))
plot_EF_Hypergeometric(EW_model_GulfCar, EV, 0, "Gulf of Carpentaria")
plot_EW_Hypergeometric(EW_model_GulfCar, "Gulf of Carpentaria")


### Plot the eigenvalues and eigenfunction for the Calgary region
### NOTE: The eigenfunction are the still the same, only the EW are different
print("Smallest eigenvalues =", min(EW_model_Calgary))
plot_EF_Hypergeometric(EW_model_Calgary, EV, 0, "Calgary Region")
plot_EW_Hypergeometric(EW_model_Calgary, "Calgary Region")
close_to_zero(EW_model_Calgary)
```

D.4. Solution to the model with a physically relevant mass density function

```
# Import area
import numpy as np
import matplotlib.pylab as plt
from scipy.sparse.linalg import spsolve
from scipy.linalg import solve_banded
import scipy

"""
In this program we compute the solutions of the the model in physical form, which is an ODE given by
b*y - [m_hat * y']' = 0, by using the Finite difference method.
Thus we are solving a BVP with y(0) = y(1) = 0.

We do this by two different approaches.
1. We do not work out the derivative
2. We fully work out the derivative, such that the Equation also contains a first order derivative

NOTE: Actual comments are given by ###, the commented code is given by a single #
"""


def FDM_Model_Physical(h, b, region):
    """
    Description
    NOTE: Version 1 which computes the solution by NOT WORKING OUT THE DERIVATIVES, into the form
          y'' + y' + y = D and solving it in the form as shown.
          We do this to make sure the discretisation matrix A will be symmetric.

```

```
    Function computes the solution to the model equation (BVP), using the Finite difference method.
    We approximate the derivatives with the Central Differences.

    The domain of this SL-Problem is: 0 < x < 1,
    With homogeneuous Dirichlet Boundary conditions: V(0) = V(1) = 0.

    The result of the FDM gives Av = RHS, where A is a SYMMETRIC TRIDIAGONAL matrix, which is of the form:
    [[M U X]
     [L M U]
     [X L M]],
    where M: Main diagonal, U: Upper diagonal, L: Lower diagonal, X: zeroes.
    From FDM we have that the matrix only contains three nonzero diagonals

    For N interior points we have that A is a N x N matrix.

    For equation i we get that the discretisation gives:
    b*y_i + (1/h^2) * (-m_(i+1/2) * y_(i+1) + (m_(i+1/2) + m_(i-1/2)) * y_i - m_(i-1/2) * y_(i-1)) = RHS_i
    The coefficients in front of the y_j are put in the diagonals of the discretisation matrix.
    After FDM we obtain the system Ay = d, we then solve this for the vector y, where d is the zero vector

    Returns a vector which is the solution of Ay = d
    This vector represents the function values of the solution to the ODE
    NOTE: This vector does not contain the values at x=0 and x=1 YET! so we need to add them!!!

    N: Number of interior points
    h: Stepsize
    m_hat_hypergeometric: Mass density function for this model
    b: Beta value
    d: RHS of the system Ay = d
    y: Vector containing the function values of the solution at the internal gridpoints
    k0: The "normalized" forcing term
    """
    ### Create the empty diagonals of the discretization matrix,
    ### where the upper and lower diagonals are one size smaller than the main diagonal
    mainDiagonal = np.zeros(N)
    upperDiagonal = np.zeros(N-1)
    lowerDiagonal = np.zeros(N-1)

    ### Forcing Term
    d = np.zeros(N)  # RHS of the sytem Ay = d after applying FDM

    ### Next we compute the diagonal entries and plug them in the diagonals
    ### NOTE: x_i = i*h, where h = 1 / (N+1)
    ### NOTE: Python start counting from i=0, so when assigning the values to the diagonals we substract 1 from the index
    for i in range(1, N+1):
        ### Take the value of x at a half index as the average of the point before and after
        x_for = ((h*i) + (h*(i+1))) / 2  # x_(i + 1/2) = (x_i + x_i+1) / 2
        x_bac = ((h*i) + (h*(i-1))) / 2  # x_(i - 1/2) = (x_i + x_i-1) / 2

        ### NOTE: When assiging values to the lower diagonals we need to subtract anothe -1 from the index,
        ### so we have for the lowers i-2, since the entries of the lower only appear after equation i=2.
        if i == 1:  # Equation with BC at x=0 (FIRST EQUATION)
            mainDiagonal[i-1] = b + (m_hat_physical(x_for) + m_hat_physical(x_bac)) / h**2  # Coeff for y_1
            upperDiagonal[i-1] = -1 * m_hat_physical(x_for) / h**2  # Coeff for y_2
            ### No lower diagonal entry in row 1 Since y_0 = y(0) = 0
        if i == N:  # Equation with BC at x = 1 (LAST EQUATION)
            mainDiagonal[i-1] = b + (m_hat_physical(x_for) + m_hat_physical(x_bac)) / h**2  # Coeff for y_N
            lowerDiagonal[i-2] = -1 * m_hat_physical(x_bac) / h**2  # Coeff for y_N-1
            ### No upper diagonal entry in row N, since y_N+1 = y(1) = 0
```

```
        else: # Equations corresponding to interior points, so all 1 < i < N
            mainDiagonal[i-1] = b + (m_hat_physical(x_for) + m_hat_physical(x_bac)) / h**2  # Coeff for y_i
            upperDiagonal[i-1] = -1 * m_hat_physical(x_for) / h**2  # Coeff for y_i+1
            lowerDiagonal[i-2] = -1 * m_hat_physical(x_bac) / h**2  # Coeff for y_i-1

        d[i-1] = k0(i*h, region)  # Assigns the values of the forcing term k0 to the RHS vector

    ### Puts the values in a vector and turns them into the diagonals of the matrx
    mainDiagonalMatrix = np.diag(mainDiagonal)
    upperDiagonalMatrix = np.diag(upperDiagonal, 1)
    lowerDiagonalMatrix = np.diag(lowerDiagonal, -1)
    A = mainDiagonalMatrix + upperDiagonalMatrix + lowerDiagonalMatrix  # Discretisation matrix

    ### Compute the condition number of the matrix (TAKES LONG)
    ### NOTE: Important for symmetric matrices we take the maximal absolute value!!! so |L|_max and not |L_max|!!!
    eigenwaarden = np.linalg.eig(A)[0]
    conditionNumber2 = np.linalg.cond(A)
    print("Condition number (other method) =", conditionNumber2)

    ### Check symmetry and Stability of the method
    ### NOTE: The norm of the inverse increases as h->0, so unsure if there exist some constant that is
    # ### always bigger than the norm of the inverse matrix.
    print("Matrix is symmetric =", scipy.linalg.issymmetric(A))
    NormInverse_A = 1 / np.min(np.abs(eigenwaarden))  # Since A is symmetric
    print("Smallest EW =", eigenwaarden[0])
    print("||A^-1|| =", NormInverse_A)
    # print("Determinant A =", np.linalg.det(A))

    ### Check if matrix is invertible, by checking if columns of matrix span R^n
    ### We do this by checking the rank of the matrix
    rank = np.linalg.matrix_rank(A)
    if rank == A.shape[0]:  # If number of pivots is equal to N of the NxN matrix, columns are independent
        print("The columns of the matrix span R^n, hence the matrix is invertible")
    else:
        print("Columns do not span R")


    ### Matrix A is symmetric and trDidiagonal so we can use more efficient function to solve Ay = d
    # y = np.linalg.solve(A, d)  # Slower method
    # y = spsolve(A, d)  # Faster method

    ### Band method (fastest method)
    diagonals = np.zeros((3, N))
    diagonals[0, 1:] = upperDiagonal
    diagonals[1, :] = mainDiagonal
    diagonals[2, :-1] = lowerDiagonal
    y = solve_banded((1,1), diagonals, d)

    # print("A =", A)
    # print("y = ", y)

    ### Add the two boundary values at x=1 and x=0 to the solution values
    y = np.concatenate(([0], y, [0]))
    return y


def m_hat_physical(x):
    """
    Description
    Coefficient which is present in the discretization using FDM:
    [-p_i+1/2 * y_i+1 + (p_i+1/2 + p_i-1/2) * y_i - p_i-1/2 * y_i-1] / h^2 = mu * y_i
    NOTE: Here we use a physical mass density function of form cosine
    """
    y = np.cos(np.pi * (2*x-1)) + 1
```

```
    return y


def k0(x, region):
    """
    Description
    The forcing term which is on the RHS of the simplified model (written in terms of s)
    These are just some trivial guesses!
    """

    if region == "Carpentaria":
        ### Forcing Term (Gulf of Carpentaria)
        # res = R_e * ((np.cos(alpha_Gulfcar))**2 + (np.sin(alpha_Gulfcar))**2 / C_Gulfcar) * forcing(x) / rho(x)  # Original
        # res = R_e * ((np.cos(alpha_Gulfcar))**2 + (np.sin(alpha_Gulfcar))**2 / C_Gulfcar) * 0  # No forcing
        # res = R_e * ((np.cos(alpha_Gulfcar))**2 + (np.sin(alpha_Gulfcar))**2 / C_Gulfcar) * 1  # Constant forcing
        res = R_e * ((np.cos(alpha_Gulfcar))**2 + (np.sin(alpha_Gulfcar))**2 / C_Gulfcar) * (5*x**2 + x + 1)  # Quadratic forcing
        # res = 5*x**2 + x + 1
    elif region == "Calgary":
        ### Forcing Term (Gulf of Carpentaria)
        # res = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 / C_Calgary) * forcing(x) / rho(x)  # Original
        # res = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 / C_Calgary) * 0  # No forcing
        # res = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 / C_Calgary) * 1  # Constant forcing
        res = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 / C_Calgary) * (5*x**2 + x + 1)  # Quadratic forcing
    return res


def RHS(x):
    """
    Description
    The right hand side of the model, which we use to check if the RHS and HomSol are orthogonal,
    in case of Calgary.
    NOTE: If you want to check the orthogonality set k0 == 0 (the in the function above) for Calgary, to obtain Hom_Sol
    """
    AAHH = R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 / C_Calgary) * (5*x**2 + x + 1)  # Quadratic k0
    # AAHH = np.zeros(N+2) + R_e * ((np.cos(alpha_Calgary))**2 + (np.sin(alpha_Calgary))**2 / C_Calgary) * 1  # Constant k0
    return AAHH


### Work space
"""
Default stepsize we have used is 1/2000
For Calgary we must use small less than 1/100 since condition number jumps up hard
"""
h = 1/2000  # Stepsize
N = int((1/h) - 1)  # Number of interior points
R_e = 10**5  # Reynolds number for the model
print("Number of interior points =", N)

### Beta values = -sigma * S * R_e
beta_GulfCar = -133 * -0.24 * 10**5 # Value of beta in the Gulf of Carpentaria, beta > 0
beta_Calgary = -8.5 * 0.77 * 10**5 # Value of beta in the Calgary region, beta < 0
# print(beta_GulfCar)
# print(beta_Calgary)


### Parameter values
sigma_Gulfcar = 133
```

```
C_Gulfcar = 0.97
S_Gulfcar = -0.24
alpha_Gulfcar = np.pi * 5 / 4  # Breeze propagating in SW-Direction

sigma_Calgary = 8.5
C_Calgary = 0.62
S_Calgary = 0.77
alpha_Calgary = np.pi / 4  # Breeze propagating in NW-Direction

### Computes the solutions with different variations of FDM
### Uncomment the line for the region for which you want to compute the solution
sol = FDM_Model_Physical(h, beta_GulfCar, "Carpentaria")  # Solves ODE in the original
    form as the model
# sol = FDM_Model_Physical(h, beta_Calgary, "Calgary")

x_as = np.linspace(0,1,N+2)
test = RHS(x_as)
plt.plot(x_as, sol)
plt.title("Horizontal velocity of the sea breeze flow in the Gulf of Carpentaria")
# plt.title("Horizontal velocity of the sea breeze flow in the Calgary region")
plt.xlabel("Height (s)")
plt.ylabel(r'$V_0(s)$')
plt.grid()
plt.show()

### Plot the mass density function
plt.plot(x_as, m_hat_physical(x_as))
plt.title("Physical mass density function")
plt.xlabel("Height (s)")
plt.ylabel(r'$\hat{m}(s)$')
plt.grid()
plt.show()

### Check if forcing function and homogeneous solution are orthogonal to each other
### RHS: k0 function
### NOTE: For this set k0==0 to obtain the homogeneous solution of the model
### NOTE: It appears that yes they are orthogonal in all cases for Calgary
### we do not look at Gulf since EW are clearly nonzero
# print("InnerProduct =", np.inner(sol, test))
```

Bibliography

[1] D.J. Abbs and W.L. Physick. Sea-breeze observations and modelling: a review. *Aust. Met. Mag.*, 41:7–19, 1992.

[2] W.O. Amrein, A.M. Hinz, and D.B. Pearson. *Sturm-Liouville Theory: past and present*. Birkhäuser Basel, 1 edition, 2005. ISBN 978-3-7643-7066-4. doi: https://doi.org/10.1007/3-7643-7359-8.

[3] R.E.A. Bouwens, W. Kranendonk, and J.P. van Lune. *Binas*. Noordhoff Uitgevers Groningen, 6th edition, 2013. ISBN 9789001817497.

[4] A Constantin and R.S. Johnson. On the propagation of nonlinear waves in the atmosphere. *Proc. R. Soc. A*, 478, 2022. doi: https://doi.org/10.1098/rspa.2021.0895.

[5] M. Ebrahimi. *Power Generation Technologies*. Academic Press, 2023. ISBN 978-0-323-95370-2. doi: https://doi.org/10.1016/C2021-0-02782-6.

[6] EUMeTrain. Meteorological physical background. URL `https://resources.eumetrain.org/satmanu/CMs/SB/navmenu.php?tab=2&page=2.0.0`.

[7] R.B. Guenther and J.W. Lee. *Sturm-Liouville problems: Theory and numerical implementation*. CRC Press, 2019. ISBN 9780429437878.

[8] R. Habermann. *Applied Partial Differential Equations with Fourier Series and Boundary Value problems*. Pearson Education Limited, 5th edition, 2013. ISBN 9781292039855.

[9] E. Hossain and R. Islam. *Drilling Engineering*. Gulf Professional Publishing, 2 edition, 2020. ISBN 978-0-12-820193-0. doi: https://doi.org/10.1016/C2019-0-00943-0.

[10] K. Marynets. A sturm-liouvlle problem arising in the atmospheric boundary-layer dynamics. *J.of Math.Fluid Mech.*, 22(41), 2020. doi: https://doi.org/10.1007/s00021-020-00507-5.

[11] K. Marynets. Sturm-liouville boundary value problem for a sea-breeze flow. *J.of Math.Fluid Mech.*, 25(6), 2023. doi: https://doi.org/10.1007/s00021-022-00747-7.

[12] E.A. Mathez and J.E. Smerdon. *Climate change: The science of global warming and our energy future*. Columbia University Press, 2018. ISBN 9780231172837.

[13] National Oceanic and Atmospheric Administration. The sea breeze. URL `https://www.noaa.gov/jetstream/ocean/sea-breeze`.

[14] J. Schmaltz. Gulf of carpentaria from modis, 2008. Figure.

[15] C. Vuik, F.J. Vermolen, M.B. van Gijzen, and M.J. Vuik. *Numerical Methods for Ordinary Differential Equations*. TUDelft Open, 2023. ISBN 978-94-6366-665-7. doi: org/10.5074/t.2023.001.

[16] J.R. Wang and Z.H. Li. Existence and stability of solutions for a sea-breeze flow model. *Monatsh Math*, 206:381–402, 2025. doi: https://doi.org/10.1007/s00605-024-02016-3.