eman ta zabal zazu
UPV EHU
ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA
50 URTE AÑOS 1968 - 2018
Biba Zientzia! Ciencia Viva

Numerical methods for delay differential equations

Final Degree Dissertation
Degree in Mathematics

Markel Irastorza Zabalegui

Supervisor:
Luis Vega González
Felipe Ponce-Vanegas

Leioa, 20 June 2024

Contents

Introduction

Differential equations have been widely used to model and explain the real world. Still, the best known differential equations, such as ordinary differential equations (ODE) or partial differential equations (PDE), do not take into account the past events that affect the present state of the system. This is why the concept of delay differential equations (DDE) was introduced.

DDEs are a type of differential equation that takes into account past events that affect the present state of the system. With DDEs, many systems can be described more accurately than with ODEs or PDEs. A good example of that is the regenerative machine tool chatter. Because of the infinite nature that delays introduce, the goal of this work is to present two possible numerical methods to study DDEs.

Chapter 1 of this work will be dedicated to present some of the most important concepts of DDEs, specifically for linear DDEs. Also, a theoretical background for the rest of the work will be given.

In Chapter 2, a possible improvement for the already well developed Semi-discretization method will be presented. With a similar procedure, in Chapter 3 a new method based on the Cayley transform will be developed: the Tustin method.

In Chapter 4, some numerical tests will be performed with the method presented in the previous chapters, to have a better understanding of the efficiency and possible problems of those.

Finally, the most important conclusions of the work will be presented, mainly using the results of the previous chapter.

It is important to note that this work is strongly related to the work in progress [7], developed by Felipe Ponce-Vanegas and myself. In that preprint, a concise and advanced analysis of the methods is made, and it is not centered around the analysis of stability. It has not been published yet, but some ideas and results from it will be used here.

Chapter 1

Important concepts

The definitions and examples presented in this introductory chapter are taken from Chapters 1 and 2 of [1], where more examples, exercises and detailed explanations can be found. For some of the definitions and examples, the original references are given.

1.1 Notation and important definitions

Notation 1. I_N is the $N \times N$ identity matrix defined in $\mathbb{C}^N$.

I is the identity operator in a Banach space.

Notation 2. When referring to an element of a matrix A, we will denote the element in the i-th row and j-th column as A_{ij}.

Notation 3. The sign $\otimes$ is the Kronecker product. During this work, we will suppose that the most basic properties of the Kronecker product are known, and we will prove some others. We will often use without further explanation $fA = f \otimes A$, for any one-dimensional function f and any matrix A of any dimension.

Notation 4. $\sigma(A)$ is the spectrum of the operator A.

Notation 5. $\delta_{i,j}$ is the Kronecker delta.

δ_x is the Dirac delta function, $\delta_x f = f(x)$, for any continuous function f.

The difference between these two should be obvious depending on the context.

Notation 6. For an interval $I \in \mathbb{R}, \mathcal{C}(I)$ denotes the space of continuous functions in I, and $\mathcal{C}_P(I)$ denotes the space of smooth, periodic functions in I.

Notation 7. If X is a Banach space, then X^* denotes the dual space of X.

Given a space X of scalar functions, the corresponding space of vector-valued functions is $X \otimes \mathbb{C}^N$. A functional $f^* \otimes v \in X^* \otimes (\mathbb{C}^N)^*$ acts on $f \otimes u \in X \otimes \mathbb{C}^N$ as $(f^* \otimes v)(f \otimes u) := f^*(f) \cdot \langle v, u \rangle$, where $\langle v, u \rangle$ is the usual inner product in $\mathbb{C}^N$.

Definition 1.1.1. Let X be a Banach space. A one parameter family $T(t)$, $0 \leq t < \infty$, of bounded linear operators from X to X is a semigroup if:

(i) $T(0) = I$.

(ii) $T(t+s) = T(t)T(s)$ for all $t, s \geq 0$.

If, in addition, $\lim_{t \to s} T_t x = T_s x, \forall x \in X, \forall s \geq 0$, then the semigroup is called a C^0-semigroup or strongly continuous semigroup.

Definition 1.1.2. For a semigroup $T(t)$, the infinitesimal generator A is defined as:

$$\begin{cases} Ax = \lim_{t \to 0} \frac{T(t)x - x}{t} = \frac{d^+ T(t)x}{dt}|_{t=0} \\ \mathcal{D}(A) = \{x \in X : \lim_{t \to 0} \frac{T(t)x - x}{t} \text{ exists}\} \end{cases}.$$

Both definitions can be found in [5], Chapter 1, Definition 1.1.

1.2 Linear DDEs

A delay differential equation (DDE) is a type of differential equation that takes into account the past events that affect the present state of the system.

Definition 1.2.1. The general form for the linear autonomous DDEs for $\mathbf{x}(t) \in \mathbb{R}^N$ is given by the following equation:

$$\dot{\mathbf{x}}(t) = \mathbf{L}(\mathbf{x}_t)$$

where $\mathbf{L} : \mathcal{C} \longrightarrow \mathbb{R}^N$ is a continuous linear functional, $\mathcal{C}$ is the Banach space of continuous functions, and the continuous function $\mathbf{x}_t$ is defined by the shift

$$\mathbf{x}_t(\theta) = x(t + \theta), \quad \theta \in [-\tau, 0]. \tag{1.1}$$

Example 1.2.1. An example of a linear DDE with only one discrete delay is the equation:

$$\dot{x}(t) = ax(t) + bx(t - \tau),$$

where $a, b \in \mathbb{R}$ and $\tau \geq 0$ is the only delay. We will refer to this equation as the Hayes equation, as its stability conditions were first presented by Hayes [2] in 1950. It is the most basic example of the type of equation that we will be studying in this work.

Example 1.2.2. Another example of a linear DDE with only one discrete delay, but this time a more complex one, is the equation:

$$\ddot{x}(t) + a_1\dot{x}(t) + a_0x(t) = b_0x(t-\tau),$$

where $a_0, a_1, b_0 \in \mathbb{R}$ and $\tau \geq 0$ is the only delay. This equation describes a dumped delay oscillator.

Example 1.2.3. An example of a linear DDE with continuously distributed delays is the equation:

$$\dot{x}(t) = ax(t) + b\int_{-\sigma}^{0} x(t+\theta)d\theta.$$

It was analyzed by Cushing [3].

Definition 1.2.2. Given the initial condition $\mathbf{x}(t) = \varphi(t)$ for the interval $[-\tau, 0]$, being $\varphi(t) \in \mathbb{R}^N$ a continuous function, the solution for the interval $[0, \tau]$ exists, is unique and is given by:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{L}(\mathbf{x}_t), & t \in [0, \tau] \\ \mathbf{x}(t) = \varphi(t), & t \in [-\tau, 0] \end{cases}.$$

Remark 1.2.1. After finding the solution $\mathbf{x}(t)$ for the interval $[0, \tau]$, the solution for the interval $[\tau, 2\tau]$ can be found by repeating the process, but using the solution for the interval $[0, \tau]$ as the initial condition. This process can be repeated as many times as needed to find the solution for the interval $[n\tau, (n+1)\tau]$, $n \in \mathbb{N}$. Thus, the solution of the system is $\mathbf{x}(t) : [-\tau, \infty] \longrightarrow \mathbb{R}^N$. This method is called the method of steps. With the method of steps, the existence and uniqueness of the solution can be assured, as it is explained in [4].

Remark 1.2.2. According to Riesz Representation Theorem, the functional $\mathbf{L}$ can be represented as:

$$\mathbf{L}(\mathbf{x}_t) = \int_{-\infty}^{0} [d\eta(\theta)]x(t+\theta) \tag{1.2}$$

where η is an $N \times N$ matrix of functions of bounded variation on $(-\infty, 0)$ and the integral is a Riemann–Stieltjes one.

Definition 1.2.3. The characteristic equation associated with a linear DDE is:

$$D(\lambda) := \det\left(\lambda I - \int_{-\infty}^{0} e^{\lambda t}\, d\eta(\theta)\right) = 0 \tag{1.3}$$

The characteristic equation can be found either by substituting the non-trivial solution $\mathbf{x}(t) = Ce^{\lambda t}$ into Equation (1.2), where C is a constant and λ is a complex number, or by using the Laplace transform.

Definition 1.2.4. The roots of the characteristic equation are called characteristic exponents.

Before going into the methods to solve these equations, one last type of DDEs will be presented: the linear periodic DDEs. Even if those are better to describe many systems, they are more complicated to analyze, even with the methods presented in this work.

Definition 1.2.5. The general form for the periodic DDEs is:

$$\dot{\mathbf{x}}(t) = \mathbf{L}(t, \mathbf{x}_t), \mathbf{L}(t+T) = \mathbf{L}(t),$$

where $\mathbf{x}_t$ is a continuous function defined by Equation (1.1), the minimal period is $T > 0$, and $\mathbf{L} : \mathbb{R} \times \mathcal{C} \longrightarrow \mathbb{R}^n$ is a continuous and linear functional in $\mathbf{x}_t$.

1.3 Our case

Among linear DDEs with constant coefficients containing only discrete delays, in this work we will only analyze the ones that contain only one discrete delay.

Definition 1.3.1. The general form for the linear DDEs containing only one discrete delay is:

$$\dot{\mathbf{x}}(t) = \mathbf{B}_0\mathbf{x}(t) + \mathbf{B}_1\mathbf{x}(t-\tau), \quad t \in [0, \tau] \tag{1.4}$$

where $\mathbf{B}_0$ and $\mathbf{B}_1$ are $N \times N$ matrices, and $\tau > 0$ is the unique discrete delay.

Remark 1.3.1. If we take Equation (1.4) and make the change of variables $t = \tau s$ and $\mathbf{x}(t) = \mathbf{y}(s)$, we get:

$$\dot{\mathbf{y}}(s) = \tau\mathbf{B}_0\mathbf{y}(s) + \tau\mathbf{B}_1\mathbf{y}(s-1), \quad s \in [0, 1].$$

So, from now on, we will refer to (1.4) as:

$$\dot{\mathbf{x}}(t) = \mathbf{B}_0\mathbf{x}(t) + \mathbf{B}_1\mathbf{x}(t-1), \quad t \in [0, 1]. \tag{1.5}$$

Even though the general formula for defining the characteristic equation was given in Definition 1.2.3, there is an easier way to define it when restricted to linear DDEs with only one discrete delay.

Definition 1.3.2. The characteristic equation associated with the DDE (1.5) is expressed as:

$$\Delta(\lambda) := \det(-\lambda I_N + \mathbf{B}_0 + \mathbf{B}_1 e^{-\lambda}) = 0, \tag{1.6}$$

where I_N is the $N \times N$ identity matrix.

Remark 1.3.2. As it was stated in Definition 1.2.4, the roots of the characteristic equation are called characteristic exponents. So, in this work, we will use the characteristic exponents to refer to the roots of $\Delta(\lambda)$.

1.4 Stability

Usually, the exact solution of the DDEs is not needed, but only the analysis of the stability of the system.

Definition 1.4.1. A system is stable if all the characteristic exponents are in the left half-plane.

In practice, only the rightmost characteristic exponent must be analyzed: if it is in the right half plane, the system is unstable; if it is in the left half-plane, the system is stable.

Remark 1.4.1. If the rightmost characteristic exponent is 0, generally the stability is undetermined. Still, in this work we will define those as stable, as they do not present an asymptotic growth.

To show the stability of a system, the most popular method is to construct an stability chart, which shows the stability of the system for different values of the parameters. Some examples of stability charts can be seen in Chapter 2 of [1]. One of the most basic ones, is the stability chart of the Hayes equation,

$$\dot{x}(t) = ax(t) + bx(t-1),$$

where $a, b \in \mathbb{R}$.

The stability chart for this equation was first presented by Hayes in 1950, and we will use it in Chapter 4 to verify the results obtained in Chapters 2 and 3, as it is the most basic case of Equation (1.5).

The procedure to build the stability chart of the Hayes equation can be found in [1]. To future comparisons, we will only take into account the stable and unstable areas, ignoring the number of unstable characteristic exponents.

1.5 Lunel's point of view

To develop Chapters 2 and 3, we will work with the differential operator. To do so, the description of the differential operator A from [4] is used.

Theorem 1.5.1. *Given*

$$\dot{\mathbf{x}}(t) = B_0\mathbf{x}(t) + B_1\mathbf{x}(t-1), \quad x_0 = \varphi, \quad \varphi \in \mathcal{C},$$

where $\mathcal{C}$ is the Banach space of continuous functions. This differential equation can be written as:

$$\frac{du}{dt} = Au, \quad u(0) = \varphi, \quad \varphi \in \mathcal{C}$$

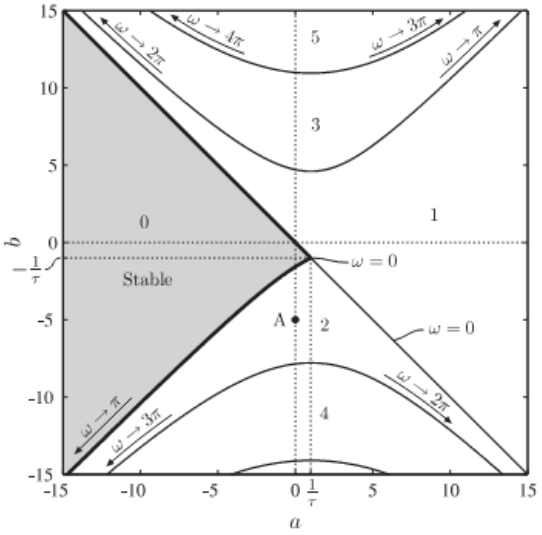

Figure 1.1: The x-axis are the values for the parameter a and the y-axis the values for the parameter b. The grey area are the values for a and b where the system is stable, and the white area the values where is unstable. It is also divided in areas according to the number of unstable characteristic exponents. The figure is taken from [1], Figure 2.1.

where A is an unbounded operator that acts on the space of continuous functions defined as:

$$\begin{cases} A\varphi = \frac{d\varphi}{d\theta} \\ \\ \mathcal{D}(A) = \{\varphi \in \mathcal{C} : A\varphi \in \mathcal{C}, \frac{d\varphi}{d\theta}(0) = B_0\varphi(0) + B_1\varphi(-1)\} \end{cases} . \tag{1.7}$$

This means that the differential operator A can be understood as the operator that sends continuous functions to continuous functions that fulfill the equation $\{\frac{d\varphi}{d\theta}(0) = B_0\varphi(0) + B_1\varphi(-1)\}$.

To use the operator A for the stability analysis, the first step is to convert it into a bounded operator, and the second step is to discretize the new operator. This is the main idea behind the Chapters 2 and 3.

1.6 Discretization

One method to discretize a bounded operator $B \in \mathcal{B}(X)$ in a Banach space X is by choosing an increasing sequence of M-dimensional subspaces such that

$$V_1 \subset \ldots \subset V_M \ldots \subset X, \text{ and } \overline{\bigcup_M V_M} = X.$$

Then, we define immersions $\iota_M : \mathbb{C}^M \longrightarrow V_M$ and projections $\pi_M : X \longrightarrow \mathbb{C}^M$ such that $\pi_M \iota_M = I_{\mathbb{C}^M}$. Also, we define the projections $P_M := \iota_M \pi_M$. Finally, we define the discretized operator as $\pi_M B \iota_M : \mathbb{C}^M \longrightarrow \mathbb{C}^M$.

After applying the discretization, we must describe the relation between the spectrum of the initial operator B and the discretized operator. To do so, we first set the relation between $\pi_M B \iota_M$ and an intermediate operator, $B_M := P_M B P_M : X \longrightarrow X$, which is $\sigma(\pi_M B \iota_M) = \sigma(B_M)$. This relationship is proven in [7]. Now, the relation between $\sigma(B_M)$ and $\sigma(B)$ was given by Osborn in [6].

It is important to note that that last relation can not be applied to the Semi-discretization method, as the operator and the discretization change as we refine the approximation. On the other hand, it is suitable for the Tustin method.

Chapter 2

Semi-discretization

2.1 Exponential function

Definition 2.1.1. The function

$$\begin{aligned} f\colon \mathbb{C} &\longrightarrow \mathbb{C} \\ z &\longmapsto e^z, \end{aligned}$$

is called the exponential function.

Theorem 2.1.1. *Let $H = \{z \in \mathbb{C} : \mathrm{Re}(z) < 0\}$ be the open left half-plane. Then, the exponential function maps the left half-plane to the unit circle.*

Proof. Let $H = \{z \in \mathbb{C} : \mathrm{Re}(z) < 0\} = \{a + bi : a, b \in \mathbb{R} \text{ and } a < 0\}$ be the open left half-plane, and $U = \{z \in \mathbb{C} : |z| < 1\}$ the unit circle. Then, for an arbitrary $z = a + bi \in H$, we have

$$\begin{aligned} e^z = e^{a+bi} = e^a e^{bi} = e^a(\cos(b) + i\sin(b)). \text{ As } a < 0,\ e^a < 1, \text{ so} \\ |e^z| = |e^a(\cos(b) + i\sin(b))| = e^a|(\cos(b) + i\sin(b))| = e^a < 1, \end{aligned}$$

what means that, $e^z \in U$. □

Corollary 2.1.2. *The exponential function sends the edge of the left half-plane to the unit circle.*

Proof. Let $\partial H = \{z \in \mathbb{C} : \mathrm{Re}(z) = 0\}$ be the edge of the left half-plane, and $\partial U = \{z \in \mathbb{C} : |z| = 1\}$ the edge of the unit circle. Then, for an arbitrary $z = bi \in \partial H$, we have $|e^z| = |e^{bi}| = |\cos(b) + i\sin(b)| = 1$, so $e^z \in \partial U$. □

Remark 2.1.1. From now on, sometimes we will use e^z to refer to the exponential function without further explanation.

2.2 Semi-discretization operator

For the Semi-discretization method, we will use the bounded operator e^{hA}.

Theorem 2.2.1. *Let $f \in \mathcal{C}([-1,0])$. Then,*

$$(e^{hA}f)(\theta) = \begin{cases} f(\theta+h), & \text{if } -1 \le \theta \le -h, \\ x(\theta+h), & \text{if } -h \le \theta \le 0, \end{cases} \tag{2.1}$$

where

$$x(\theta) = e^{B_0\theta}f(0) + \int_0^\theta e^{B_0(\theta-s)}B_1 f(s-1)\,ds, \text{ for } \theta \in [0,1].$$

Proof. We only sketch a proof, as it is beyond the reach of this work. Let $\{T(h)\}_{h\ge 0}$ be the semi-group defined by Equation (2.1). If A is the infinitesimal generator of $T(h)$, then by Theorem 2.6 in [5] about the uniqueness of the associated semi-group, we would conclude that $T(h) = e^{hA}$.

Hence, we must prove that

$$\lim_{h\to 0^+} \left\| \frac{T(h)f - f}{h} - Af \right\|_{L^\infty} = 0, \quad \text{for all } f \in \mathcal{D}(A). \tag{2.2}$$

This follows from the uniform continuity of f', and the boundary condition $f'(0) = B_0 f(0) + B_1 f(-1)$. □

2.3 Eigenvalues of the semi-discretization operator

As it has been already explained, the goal of the semi-discretization is to bound the spectral values of the operator A to make easier to decide if the system is stable or not. After applying the exponential to the operator A, we must set the relation between the spectral values of A and e^{hA}.

Theorem 2.3.1. *Let $\sigma(A)$ be the spectrum of the operator A defined in Theorem 1.7. Then,*

$$\sigma(e^{hA})\backslash\{0\} = e^{h\sigma(A)}.$$

Remark 2.3.1. Using the Spectral Mapping Theorem, we would have expected that the relation were $\sigma(e^{hA}) = e^{h\sigma(A)}$. But, it has already been proved that that relation is not always true. Both the counterexample for the first supposition and the proof of Theorem can be found in [5], chapter 2, Example 2.1 and Theorem 2.4, respectively.

2.4 Discretization of the semi-discretization operator

Now, the discretization of the operator e^{hA} will be presented. The first step is to define the space, which will be $X := \mathcal{C}([-1,0];\mathbb{C}^N)$, the space of continuous functions from $[-1,0]$ to $\mathbb{C}^N$. Next, we chose the subspaces $V_h \otimes \mathbb{C}^N$, where $V_h = \langle v_0, ..., v_{M-1} \rangle$, for $h = \frac{1}{M-1}$, is the M-dimensional subspace generated by the linear splines defined as:

$$v_0(\theta) = \begin{cases} -\frac{\theta+1}{h} + 1, & \text{if } -1 \leq \theta \leq -1+h \\ 0, & \text{if } -1+h < \theta < 0 \end{cases}, \tag{2.3}$$

$$v_{M-1}(\theta) = \begin{cases} 0, & \text{if } -1 < \theta < -h \\ \frac{\theta+1}{h} - M + 2, & \text{if } -h \leq \theta \leq 0 \end{cases}, \tag{2.4}$$

$$v_m(\theta) = \begin{cases} 0, & \text{if } -1 < \theta < -1+(m-1)h \\ -\frac{\theta+1}{h} - m + 1, & \text{if } -1+(m-1)h \leq \theta \leq -1+mh \\ \frac{\theta+1}{h} + m + 1, & \text{if } -1+mh \leq \theta \leq -1+(m+1)h \\ 0, & \text{if } -1+(m+1)h < \theta < 0 \end{cases}, \tag{2.5}$$

for $m = 1, ..., M-2$.

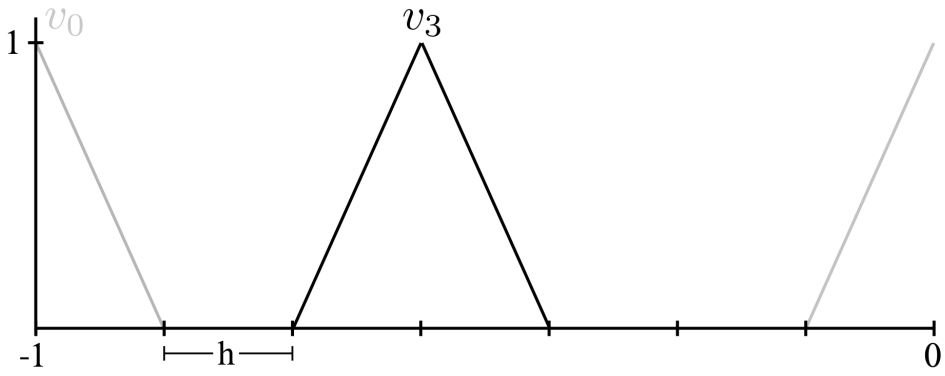

Figure 2.1: Example of the linear splines. v_3 is coloured in black, and v_0 and v_{M-1} in grey.

The basis of X is $\{v_i \otimes \mathbf{e}_j\}$, for $i = 0, ..., M-1$, $j = 1, ..., N$, where $\mathbf{e}_j$ is the j-th vector of the canonical basis of $\mathbb{C}^N$, what means that, for a function

f,

$$f \otimes \mathbf{e}_j = \begin{bmatrix} 0 \\ ... \\ f \\ ... \\ 0 \end{bmatrix}.$$

Also, a dual basis must be selected, $\{v^n \otimes \mathbf{e}^i\}$, for $i = 0, ..., M-1$, $j = 1, ..., N$, with $v^n \in (\mathcal{C}([-1, 0]))^*$ such that $v^n(v_m) = \delta_{n,m}$, and $\mathbf{e}^i \in (\mathbb{C}^N)^*$ such that $\mathbf{e}^i(\mathbf{e}_j) = \delta_{i,j}$. A possible basis is $v^n = \delta_{-1+nh}$, where $\delta_a(f) = f(a), a \in [-1, 0]$ is the Dirac delta function in a.

In this case, the pair ι_h, π_h is defined as:

$$\iota_h(z_{0,1}, \ldots, z_{0,N}, z_{1,1}, \ldots, z_{1,N}, \ldots, z_{M-1,1}, \ldots, z_{M-1,N}) = \sum_{i,j} z_{i,j}(v_i \otimes \mathbf{e}_j), \tag{2.6}$$

$$\pi_h f = ((v^0 \otimes \mathbf{e}^1)f, \ldots, (v^{M-1} \otimes \mathbf{e}^N)f). \tag{2.7}$$

Example 2.4.1. Let ι_h, π_h be the pair defined in Equations (2.6) and (2.7). Then, for the case $N = 1$, the projection $P_h = \iota_h \pi_h$ acts on $f \in \mathcal{C}([-1, 0])$ as:

$$P_h f = \sum_i f(-1 + ih)v_i.$$

Note that $P_h f \in \mathcal{C}([-1, 0])$.

Proof. Using the definition, and considering that $v^i = \delta_{-1+ih}$,

$$\begin{aligned} P_h f = \iota_h \pi_h f &= \iota_h(v^0 f, \ldots, v^i f, \ldots, v^{M-1} f) \\ &= \iota_h(f(-1), \ldots, f(-1 + ih), \ldots, f(0))) \\ &= \sum_i f(-1 + ih)v_i \\ &= \sum_i f(-1 + ih)v_i. \end{aligned}$$

□

Now, the discretization of the operator e^{hA} can be computed by using the basis and the dual basis:

$$S^h_{(n,i),(m,j)} = (v^n \otimes \mathbf{e}^i)\Big(e^{hA}(v_m \otimes \mathbf{e}_j)\Big). \tag{2.8}$$

First, let us see how e^{hA} acts on the splines previously defined:

$$(e^{hA}v_m)(\theta) = \begin{cases} v_{m-1}(\theta), & \text{if } -1 \leq \theta < -h \\ e^{B_0(\theta+h)}v_m(0) + \int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1 v_m(s-1)ds, & \text{if } -h \leq \theta \leq 0 \end{cases} \tag{2.9}$$

Remark 2.4.1. In the case $\theta = -h$, both cases can be considered, as we will see in Remark 2.4.2.

Considering where Equation (2.9) is zero, we can separate it in four cases:

$$m = 0 \Rightarrow (e^{hA}v_0)(\theta) = \int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1 v_0(s-1)ds, \quad -h \leq \theta \leq 0.$$

$$m = 1 \Rightarrow (e^{hA}v_1)(\theta) = \begin{cases} v_0(\theta), & \text{if } -1 \leq \theta \leq -h \\ \int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1 v_1(s-1)ds, & \text{if } -h \leq \theta \leq 0 \end{cases}$$

$$1 < m < M-1 \Rightarrow (e^{hA}v_m)(\theta) = v_{m-1}(\theta), \quad -1 \leq \theta \leq -h.$$

$$m = M-1 \Rightarrow (e^{hA}v_{M-1})(\theta) = \begin{cases} v_{M-2}(\theta), & \text{if } -1 \leq \theta \leq -h \\ v_{M-1}(0)e^{B_0(\theta+h)}, & \text{if } -h \leq \theta \leq 0 \end{cases}$$

But, the elements of our basis are expressed as $v_m \otimes \mathbf{e}_j$, so we must adapt those four situations to the elements on the basis.

(i) If $m = 0$,

$$(e^{hA}(v_0 \otimes \mathbf{e}_j))(\theta) = \int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1(v_0 \otimes \mathbf{e}_j)(s-1)ds, \ -h \leq \theta \leq 0.$$

(ii) If $m = 1$,

$$(e^{hA}(v_1 \otimes \mathbf{e}_j))(\theta) = \begin{cases} (v_1 \otimes \mathbf{e}_j)(\theta+h), & \text{if } -1 \leq \theta \leq -h \\ \int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1(v_1 \otimes \mathbf{e}_j)(s-1)ds, & \text{if } -h \leq \theta \leq 0 \end{cases}$$

(iii) If $1 < m < M-1$,

$$(e^{hA}(v_m \otimes \mathbf{e}_j))(\theta) = v_{m-1}(\theta), -1 \leq \theta \leq -h.$$

(iv) If $m = M-1$,

$$(e^{hA}(v_{M-1} \otimes \mathbf{e}_j))(\theta) = \begin{cases} (v_{M-1} \otimes \mathbf{e}_j)(\theta+h), & \text{if } -1 \leq \theta \leq -h \\ (v_{M-1} \otimes \mathbf{e}_j)(0)e^{B_0(\theta+h)}, & \text{if } -h \leq \theta \leq 0 \end{cases}$$

Theorem 2.4.1. *Let $A(\theta)$ be a complex matrix that changes with θ. Then,*

$$A(\theta)\mathbf{e}_j = a_{1j}(\theta) \otimes \mathbf{e}_1 + ... + a_{Nj}(\theta) \otimes \mathbf{e}_N.$$

Proof. $A(\theta)$ can be written as $\begin{bmatrix} a_{11}(\theta) & \cdots & a_{1j}(\theta) & \cdots & a_{N1}(\theta) \\ \vdots & & \vdots & & \vdots \\ a_{N1}(\theta) & \cdots & a_{Nj}(\theta) & \cdots & a_{NN}(\theta) \end{bmatrix}$, so

$$A(\theta)\mathbf{e}_j = \begin{bmatrix} a_{11}(\theta) & \cdots & a_{1j}(\theta) & \cdots & a_{N1}(\theta) \\ \vdots & & \vdots & & \vdots \\ a_{N1}(\theta) & \cdots & a_{Nj}(\theta) & \cdots & a_{NN}(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} a_{1j}(\theta) \\ \vdots \\ a_{Nj}(\theta) \end{bmatrix}.$$

That last expression can be rewritten according to the basis of $\mathbb{C}^N$, $\{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$, as $a_{1j}(\theta)\mathbf{e}_1 + \ldots + a_{Nj}(\theta)\mathbf{e}_N$. From the properties of the Kronecker product, as $a_{i,j}(\theta) \in \mathbb{C}, \forall i \in \{1, \ldots, N\}$,

$$a_{1j}(\theta)\mathbf{e}_1 + \ldots + a_{Nj}(\theta)\mathbf{e}_N = a_{1j}(\theta) \otimes \mathbf{e}_1 + \ldots + a_{Nj}(\theta) \otimes \mathbf{e}_N.$$

□

Lemma 2.4.2. *Let $v_m \otimes \mathbf{e}_j$ be an arbitrary element of the basis of the space $V_h \otimes \mathbb{C}^N$. Then,*

(i)

$$(v_m \otimes \mathbf{e}_j)(\theta + h) = (v_{m-1}(\theta) \otimes \mathbf{e}_j), \quad \textit{for } 1 < m < M.$$

(ii)

$$e^{B_0(\theta+h)}(v_m \otimes \mathbf{e}_j)(0) = \left(v_m(0)e_{1j}^{B_0(\theta+h)} \otimes \mathbf{e}_1\right) + \cdots + \left(v_m(0)e_{Nj}^{B_0(\theta+h)} \otimes \mathbf{e}_N\right).$$

(iii)

$$\int_0^{\theta+h} e^{B_0(\theta+h-s)} B_1(v_m \otimes \mathbf{e}_j)(s-1)\, ds = \left[\left(e^{B_0(\theta+h)} \int_0^{\theta+h} e^{-B_0 s} v_m(s-1) ds B_1\right)_{1j} \otimes \mathbf{e}_1\right] + \cdots + \left[\left(e^{B_0(\theta+h)} \int_0^{\theta+h} e^{-B_0 s} v_m(s-1)\, ds B_1\right)_{Nj} \otimes \mathbf{e}_N\right].$$

Proof. Let $v_m \otimes \mathbf{e}_j$ be an arbitrary element of the basis. Then, by using the properties of the Kronecker product, we will prove (i), (ii) and (iii).

(i)

$$(v_m \otimes \mathbf{e}_j)(\theta + h) = \begin{bmatrix} 0 \\ \vdots \\ v_m(\theta + h) \\ \vdots \\ 0 \end{bmatrix} = v_m(\theta + h)\mathbf{e}_j$$
$$= v_{m-1}(\theta)\mathbf{e}_j = (v_{m-1}(\theta) \otimes \mathbf{e}_j).$$

(ii) First,

$$(v_m \otimes \mathbf{e}_j)(0) = \begin{bmatrix} 0 \\ \vdots \\ v_m(0) \\ \vdots \\ 0 \end{bmatrix} = v_m(0)\mathbf{e}_j = (v_m(0) \otimes \mathbf{e}_j)$$

Hence,

$$e^{B_0(\theta+h)}(v_m \otimes \mathbf{e}_j)(0) = e^{B_0(\theta+h)}(v_m(0) \otimes \mathbf{e}_j) = v_m(0)e^{B_0(\theta+h)}\mathbf{e}_j.$$

Then, by using Theorem 2.4.1,

$$v_m(0)e^{B_0(\theta+h)}\mathbf{e}_j = \left(v_m(0)e_{1j}^{B_0(\theta+h)}\right) \otimes \mathbf{e}_1 + \ldots + \left(v_m(0)e_{Nj}^{B_0(\theta+h)}\right) \otimes \mathbf{e}_N.$$

(iii) It can be proven following the same procedure as in (ii).

□

The last step to compute the elements of S^h is to apply the elements of the dual basis to each $e^{hA}(v_m \otimes \mathbf{e}_j)$. To do so, we will use the following lemma that complements the previous one.

Lemma 2.4.3. *Let $\{v^n \otimes \mathbf{e}^i\}$ be the dual basis of $V_h \otimes \mathbb{C}^N$. Then,*

(i)

$$(v^n \otimes \mathbf{e}^i)(v_{m-1} \otimes \mathbf{e}_j) = \delta_{n,m-1}\delta_{i,j}.$$

(ii)

$$(v^n \otimes \mathbf{e}^i)\left[\left(v_m(0)e_{1j}^{B_0(\theta+h)} \otimes \mathbf{e}_1\right) + \ldots + \left(v_m(0)e_{Nj}^{B_0(\theta+h)} \otimes \mathbf{e}_N\right)\right]$$
$$= \delta_{-1+nh}\left(v_m(0)e_{i,j}^{B_0(\theta+h)}\right).$$

(iii)

$$\begin{aligned}&(v^n \otimes \mathbf{e}^i)\Big[\Big(e^{B_0(\theta+h)}\int_0^{\theta+h} e^{-B_0 s}v_m(s-1)dsB_1\Big)_{1j}\otimes \mathbf{e}_1\Big]\\ &+\ldots+\Big[\Big(e^{B_0(\theta+h)}\int_0^{\theta+h} e^{-B_0 s}v_m(s-1)dsB_1\Big)_{Nj}\otimes \mathbf{e}_N\Big]\\ &=\delta_{-1+nh}\Big[\Big(e^{B_0(\theta+h)}\int_0^{\theta+h} e^{-B_0 s}v_m(s-1)dsB_1\Big)_{i,j}\Big].\end{aligned}$$

Proof. (i)

$$\begin{aligned}(v^n \otimes \mathbf{e}^i)(v_{m-1}\otimes \mathbf{e}_j) &= v^n(v_{m-1})\mathbf{e}^i(\mathbf{e}_j)\\ &= \delta_{-1+nh}(v_{m-1}(\theta))\delta_{i,j}=\delta_{n,m-1}\delta_{i,j}.\end{aligned}$$

(ii)

$$\begin{aligned}&(v^n \otimes \mathbf{e}^i)\Big[\Big(v_m(0)e_{1j}^{B_0(\theta+h)}\otimes \mathbf{e}_1\Big)+\ldots+\Big(v_m(0)e_{Nj}^{B_0(\theta+h)}\otimes \mathbf{e}_N\Big)\Big]\\ &=(v^n \otimes \mathbf{e}^i)\Big[\Big(v_m(0)e_{1j}^{B_0(\theta+h)}\otimes \mathbf{e}_1\Big)\\ &\quad+\ldots+(v^n \otimes \mathbf{e}^i)\Big(v_m(0)e_{Nj}^{B_0(\theta+h)}\otimes \mathbf{e}_N\Big]\\ &=\Big[\delta_{-1+nh}\Big(v_m(0)e_{1j}^{B_0(\theta+h)}\Big)\delta_{i,1}\Big]+\ldots+\Big[\delta_{-1+nh}\Big(v_m(0)e_{Nj}^{B_0(\theta+h)}\Big)\delta_{i,N}\Big]\\ &=\delta_{-1+nh}\Big(v_m(0)e_{i,j}^{B_0(\theta+h)}\Big).\end{aligned}$$

(iii) Same procedure as in (ii).

□

Now, we can compute the elements of S^h, analyzing the four possibilities exposed previously, combined with the results obtained in Lemmas 2.4.2 and 2.4.3. Also, we will consider the bounds of θ in terms of n:

- If $-1 \leq \theta \leq -h$, then, when $v^n = \delta_{-1+nh}$ is applied to a function that depends of θ, $-1 \leq -1+nh \leq -h$, and with a straight forward development, $0 \leq n \leq M-2$, $n \in \mathbb{Z}^+$.
- If $-h \leq \theta \leq 0$, then $-h \leq -1+nh \leq 0 \Rightarrow M-2 \leq n \leq M-1$, $n \in \mathbb{Z}^+$.

Remark 2.4.2. In the case of $n = M-2$, it does not really matter if we consider $-1 \leq \theta \leq -h$ or $-h \leq \theta \leq 0$.

- In the first case, as $m \leq M-1$, the only case where $v^n(v_{m-1}) \neq 0$ is when $m = M-1$, in which case $v^n(v_{M-2}(\theta)) = \delta_{i,j}$.

- In the second case,

  $v^n(v_{M-1}(0)e^{B_0(\theta+h)}) \neq 0$ if and only if $m = M-1$, in which case
  $v^n(v_{M-1}(0)e^{B_0(\theta+h)}_{i,j}) = \delta_{i,j}$

  and

  $$\Big(e^{B_0(-1+(M-1)h)}\int_0^{-1+(M-1)h} e^{-B_0 s}v_0(s-1)dsB_1\Big)_{i,j} = 0,$$

  because $h = \frac{1}{M-1}$, what means that $-1+(M-1)h = 0$ and the bounds of the integral are 0 and 0, so the integral is 0.

So, to make computations easier, from now on we will evaluate $n = M-2$ in the first align.

(i) If $m = 0$, $(v^n \otimes \mathbf{e}^i)(e^{hA}(v_0 \otimes \mathbf{e}_j)(\theta))$

$$\begin{aligned}
&= (v^n \otimes \mathbf{e}^i)\Big(\int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1(v_0 \otimes \mathbf{e}_j)(s-1)ds\Big)\\
&= \delta_{-1+nh}\Big[\Big(e^{B_0(\theta+h)}\int_0^{\theta+h} e^{-B_0 s}v_0(s-1)dsB_1\Big)_{i,j}\Big]\\
&= \Big(e^{B_0(-1+(n+1)h)}\int_0^{-1+(n+1)h} e^{-B_0 s}v_0(s-1)dsB_1\Big)_{i,j},
\end{aligned}$$

for $n = M-1$.

$$\begin{aligned}
&\Big(e^{B_0(-1+Mh)}\int_0^{-1+Mh} e^{-B_0 s}v_0(s-1)dsB_1\Big)_{i,j}\\
&= \Big(e^{B_0 h}\int_0^{h} e^{-B_0 s}v_0(s-1)dsB_1\Big)_{i,j}.
\end{aligned}$$

As $s \in [0,h]$, $s-1 \in [-1,-1+h]$, so $v_0(s-1) = 1-\frac{s}{h}$, for all $s \in [0,h]$. Hence,

$$\Big(e^{B_0 h}\int_0^h e^{-B_0 s}v_0(s-1)dsB_1\Big)_{i,j} = \Big(e^{B_0 h}\int_0^h e^{-B_0 s}\Big(1-\frac{s}{h}\Big)dsB_1\Big)_{i,j}.$$

Let us first solve the integral independently, and then we will multiply by $e^{B_0(h)}$ from the left and B_1 from the right.

$$\int_0^h e^{-B_0 s}\Big(1-\frac{s}{h}\Big)ds = \int_0^h e^{-B_0 s}ds - \int_0^h e^{-B_0 s}\frac{s}{h}ds.$$

We will use the primitive function of $e^{-B_0 s}$, $\frac{d}{ds}e^{-B_0 s} = -B_0^{-1}e^{-B_0 s}$. By substituting the primitive function and applying the limits, we

get the solution for the first integral. For the second, first we must integrate it by parts and use the primitive function twice. Finally, we get

$$e^{B_0 h}\int_0^h e^{-B_0 s}\Big(1-\frac{s}{h}\Big)\,ds B_1 = \Big[-B_0^{-1}\Big(\frac{1}{h}B_0^{-1}\Big[I_N - e^{B_0 h}\Big] + e^{B_0 h}\Big)B_1\Big].$$

So,

$$\begin{aligned}&(v^n\otimes \mathbf{e}^i)\Big(e^{hA}(v_0\otimes \mathbf{e}_j)(\theta)\Big) = 0, \forall n\in 0,...,M-2,\\ &(v^{M-1}\otimes \mathbf{e}^i)\Big(e^{hA}(v_0\otimes \mathbf{e}_j)(\theta)\Big)\\ &= \Big[-B_0^{-1}\Big(\frac{1}{h}B_0^{-1}\Big[I_N - e^{B_0 h}\Big] + e^{B_0 h}\Big)B_1\Big]_{i,j}.\end{aligned}$$

(ii) If $m=1$, $(v^n\otimes \mathbf{e}^i)(e^{hA}(v_1\otimes \mathbf{e}_j)(\theta))$

$$= \begin{cases} (v^n\otimes \mathbf{e}^i)(v_0(\theta)\otimes \mathbf{e}_j), \text{ for } n=0,...,M-2\\ (v^n\otimes \mathbf{e}^i)\Big(\int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1(v_1\otimes \mathbf{e}_j)(s-1)ds\Big), \text{ for } n=M-1\end{cases}$$

Let us analyze the two cases:

- $(v^n\otimes \mathbf{e}^i)(v_0(\theta)\otimes \mathbf{e}_j) = \delta_{n,0}\delta_{i,j} = \delta_{i,j}$ if and only if $n=0$.
- Following the same procedure as for $m=0$, with the difference that $v_1(s-1)=\frac{s}{h}, s\in[0,h]$,

$$\begin{aligned}(v^n\otimes \mathbf{e}^i)\Big(\int_0^{\theta+h} e^{B_0(\theta+h-s)}B_1(v_1\otimes \mathbf{e}_j)(s-1)ds\Big)\\ = \Big[-B_0^{-1}\Big(I_N + \frac{1}{h}B_0^{-1} - \frac{1}{h}B_0^{-1}e^{B_0 h}\Big)\Big]_{i,j}.\end{aligned}$$

So,

$$\begin{aligned}&(v^0\otimes \mathbf{e}^i)\Big(e^{hA}(v_1\otimes \mathbf{e}_j)(\theta)\Big) = \delta_{i,j},\\ &(v^n\otimes \mathbf{e}^i)\Big(e^{hA}(v_1\otimes \mathbf{e}_j)(\theta)\Big) = 0, \forall n\in 1,...,M-2,\\ &(v^{M-1}\otimes \mathbf{e}^i)\Big(e^{hA}(v_1\otimes \mathbf{e}_j)(\theta)\Big)\\ &= \Big[-B_0^{-1}\Big(\frac{1}{h}B_0^{-1}\Big[I_N - e^{B_0 h}\Big] + e^{B_0 h}\Big)B_1\Big]_{i,j}.\end{aligned}$$

(iii) If $1<m<M-1$, $(v^n\otimes \mathbf{e}^i)(v_{m-1}(\theta)\otimes \mathbf{e}_j) = \delta_{n,m-1}\delta_{i,j}$.
So, for $\forall m\in 2,...,M-2$,

$$\begin{aligned}&(v^{m-1}\otimes \mathbf{e}^i)\Big(e^{hA}(v_m\otimes \mathbf{e}_j)(\theta)\Big) = \delta_{i,j},\\ &(v^n\otimes \mathbf{e}^i)\Big(e^{hA}(v_1\otimes \mathbf{e}_j)(\theta)\Big) = 0, \forall n\neq m-1.\end{aligned}$$

(iv) If $m = M-1$, $(v^n \otimes \mathbf{e}^i)\Big(e^{hA}(v_{M-1} \otimes \mathbf{e}_j)(\theta)\Big)$

$$= \begin{cases} (v^n \otimes \mathbf{e}^i)(v_{M-2}(\theta) \otimes \mathbf{e}_j), & \text{for } n = 0, ..., M-2 \\ (v^n \otimes \mathbf{e}^i)\Big(e^{B_0(\theta+h)}(v_{M-1}(0) \otimes \mathbf{e}_j)\Big), & \text{for } n = M-1 \end{cases}$$

- $(v^n \otimes \mathbf{e}^i)(v_{M-2}(\theta) \otimes \mathbf{e}_j) = \delta_{n,M-2}\delta_{i,j} = \delta_{i,j}$ if and only if $n = M-2$.
- $(v^{M-1} \otimes \mathbf{e}^i)\Big(e^{B_0(\theta+h)}(v_{M-1}(0) \otimes \mathbf{e}_j)\Big) = \delta_0\Big(v_{M-1}(0)e_{i,j}^{B_0(\theta+h)}\Big) = e_{i,j}^{B_0 h}$.

So,

$$(v^{M-2} \otimes \mathbf{e}^i)\Big(e^{hA}(v_{M-1} \otimes \mathbf{e}_j)(\theta)\Big) = \delta_{i,j},$$
$$(v^{M-1} \otimes \mathbf{e}^i)\Big(e^{hA}(v_{M-1} \otimes \mathbf{e}_j)(\theta)\Big) = e_{i,j}^{B_0 h}.$$

After computing all the possibilities, we can conclude that all the elements of the matrix S^h are zero except for:

$$\begin{aligned} S^M_{(M-1,i),(0,j)} &= \Big[-B_0^{-1}\Big(\frac{1}{h}B_0^{-1}\Big[I_N - e^{B_0 h}\Big] + e^{B_0 h}\Big)B_1\Big]_{i,j}, \\ S^M_{(0,i),(1,j)} &= \delta_{i,j}, \\ S^M_{(M-1,i),(1,j)} &= \Big[-B_0^{-1}\Big(I_N + \frac{1}{h}B_0^{-1} - \frac{1}{h}B_0^{-1}e^{B_0 h}\Big)B_1\Big]_{i,j}, \\ S^M_{(m-1,i),(m,j)} &= \delta_{i,j}, \text{ for } 1 < m < M-1 \\ S^M_{(M-2,i),(M-1,j)} &= \delta_{i,j}, \\ S^M_{(M-1,i),(M-1,j)} &= e_{i,j}^{B_0 h}. \end{aligned}$$

2.5 Original semi-discretization

The original idea of semi-discretization method, the discretization is performed in the same way, but the elements of the basis are $\{\tilde{v}_i \otimes \mathbf{e}_j\}$, for $i = 0, \ldots, M-1$ and $j = 1, \ldots, N$, where, for $i = 0, \ldots, M-2$,

$$\tilde{v}_i(\theta) = \begin{cases} 0, & \text{if } -1 \le \theta < -1+ih \text{ or } -1+(i+1)h \le \theta \le 0 \\ 1, & \text{if } -1+ih \le \theta < -1+(i+1)h \end{cases}, \text{ and}$$

$$\tilde{v}_{M-1}(\theta) = \begin{cases} 0, & \text{if } -1 \le \theta < 0 \\ 1, & \text{if } \theta = 0 \end{cases}.$$

With that basis, and being the dual basis the same as in the previous section, the elements of the matrix $\tilde{S}^h$ are computed as

$$\tilde{S}^h_{(n,i),(m,j)} = (v^n \otimes \mathbf{e}^i)\Big(e^{hA}(\tilde{v}_m \otimes \mathbf{e}_j)\Big).$$

Following the same procedure as for the matrix S^h, we can compute each element of $\tilde{S}^h$. The results are, that all entries of $\tilde{S}^h$ are zero except for:

$$\begin{aligned}
\tilde{S}^M_{(M-1,i),(0,j)} &= \left[\left(I_N - e^{B_0 h}\right) B_0^{-1} B_1\right]_{i,j}, \\
\tilde{S}^M_{(m-1,i),(m,j)} &= \delta_{i,j}, \text{ for } 0 < m < M \\
\tilde{S}^M_{(M-1,i),(M-1,j)} &= [e^{B_0 h}]_{i,j}.
\end{aligned}$$

Notation. To prevent future confusions, and because the first semi-discretization method that we presented is better grounded theoretically, we will call the first method the *improved Semi-discretization method*, and its discretized matrix S^h; and we will reference the second method just as the *Semi-discretization method*, and its discretized matrix $\tilde{S}^h$.

Chapter 3

The Tustin method

3.1 Cayley transform

Definition 3.1.1. The function

$$\begin{aligned} \varphi_{\mathcal{C}} \colon \mathbb{C} &\longrightarrow \mathbb{C} \\ z &\longmapsto \frac{1+z}{1-z}, \end{aligned}$$

is called the Cayley transform.

In this section we will describe a new numerical method to analyze the stability of a system. We called this method "The Tustin method". To present this new method, we will follow a similar procedure as for the semi-discretization method.

Theorem 3.1.1. *Let $\varphi_{\mathcal{C}}$ be the Cayley transform. Then, $\varphi_{\mathcal{C}}$ transforms the left-half plane into the unit disk.*

Proof. Let $H = \{z \in \mathbb{C} : \mathrm{Re}(z) < 0\} = \{a + bi : a, b \in \mathbb{R} \text{ and } a < 0\}$ be the open left half-plane, and $U = \{z \in \mathbb{C} : |z| < 1\}$ the unit circle. Then, for an arbitrary $z = a + bi \in H$, we have

$$\varphi_{\mathcal{C}}(z) = \varphi_{\mathcal{C}}(a + bi) = \frac{1 + a + bi}{1 - a - bi}.$$

Using some basic properties of the complex numbers and their module,

$$|\varphi_{\mathcal{C}}(z)| = \frac{|1 + a + bi|}{|1 - a - bi|} = \frac{\sqrt{(1+a)^2 + b^2}}{\sqrt{(1-a)^2 + b^2}} = \frac{\sqrt{1 + 2a + a^2 + b^2}}{\sqrt{1 - 2a + a^2 + b^2}} < 1,$$

if and only if $1+2a+a^2+b^2 < 1-2a+a^2+b^2$, and simplifying the inequality, $a < 0$, which is true by hypothesis.
So, $\varphi_{\mathcal{C}}(z) \in U$. □

Corollary 3.1.2. *Let $\varphi_{\mathcal{C}}$ be the Cayley transform. Then, $\varphi_{\mathcal{C}}$ transforms the edge of the left-half plane into the unit circle.*

Proof. Let $\partial H = \{z \in \mathbb{C} : \mathrm{Re}(z) = 0\}$ be the edge of the left half-plane, and $\partial U = \{z \in \mathbb{C} : |z| = 1\}$ the unit circle. Then, for an arbitrary $z = bi \in \partial H$, we have $|\varphi_{\mathcal{C}}(z)| = |\varphi_{\mathcal{C}}(bi)| = \frac{|1+bi|}{|1-bi|} = 1$, so $\varphi_{\mathcal{C}}(z) \in \partial U$. □

3.2 Tustin operator

Definition 3.2.1. Let A be the operator defined in Equation (1.7), and $\varphi_{\mathcal{C}}$ the Cayley transform. Then, the Tustin operator $\varphi_{\mathcal{C}}(A)$ is defined by

$$\varphi_{\mathcal{C}}(A) = (I + A)(I - A)^{-1},$$

where I is the identity operator.

Theorem 3.2.1. *Let A be the operator defined in Equation (1.7), and $\varphi_{\mathcal{C}}(A)$ the Tustin operator. Then, the action of the operator $\varphi_{\mathcal{C}}(A)$ over a function $f(\theta) \in \mathcal{C}$ is given by:*

$$(\varphi_{\mathcal{C}}(A))(f(\theta)) = -f(\theta) + 2C(f)e^{\theta} - 2\int_{-1}^{\theta} e^{\theta - s} f(s)ds, \tag{3.1}$$

where $C(f) = (I - B_0 - B_1 e^{-1})^{-1}\Big[(I - B_0)\int_{-1}^{0} e^{-s} f(s)ds + f(0)\Big]$.

Proof. The proof is divided in three steps. First, we will compute $((I - A)^{-1})(f(\theta))$; second, we will find the constant $C(f)$; and, finally, we will apply $(I + A)$ to $((I - A)^{-1})(f(\theta))$.

Step 1: Our goal is to find a function $g(\theta) \in \mathcal{D}(A)$ such that $(I - A)g(\theta) = f(\theta)$. By expanding that expression, we get an ODE that we must solve:

$$(I - A)(g(\theta)) = (g(\theta)) - \frac{dg}{d\theta}(\theta) = g(\theta) - g'(\theta) = f(\theta), \quad \theta \in [-1, 0].$$

First, let us solve the homogeneous equation:

$$g(\theta) - g'(\theta) = 0 \Rightarrow g_h(\theta) = Ce^{\theta}, \quad C \in \mathbb{R},$$

where C is a constant.
For the non-homogeneous solution, we have to substitute the particular solution $g_p(\theta) = C(\theta)e^{\theta}$ into the differential equation and solve it for $C(\theta)$,

$$g_p(\theta) - g_p'(\theta) = C(\theta) - C'(\theta)e^{\theta} - C(\theta) = C'(\theta)e^{\theta} = f(\theta).$$

By isolating C' and solving it, we get

$$C(\theta) = -\int_{-1}^{\theta} e^{-s} f(s)ds, \text{ and } g_p(\theta) = -\int_{-1}^{\theta} e^{\theta - s} f(s)ds.$$

Finally, the general solution is:

$$g(\theta) = g_h(\theta) + g_p(\theta) = Ce^{\theta} - \int_{-1}^{\theta} e^{\theta - s} f(s) ds.$$

Step 2: For $g(\theta)$ to be in $\mathcal{D}(A)$, g must fulfill the condition

$$\frac{dg}{d\theta}(0) = B_0 g(0) + B_1 g(-1). \tag{3.2}$$

The derivative of $g(\theta)$, by using the Fundamental Theorem of Calculus, is

$$\frac{dg}{d\theta}(\theta) = g'(\theta) = Ce^{\theta} - \int_{-1}^{\theta} e^{\theta - s} f(s) ds - f(\theta),$$
$$\frac{dg}{d\theta}(0) = C - \int_{-1}^{0} e^{-s} f(s) ds - f(0);$$

$g(0) = C - \int_{-1}^{0} e^{-s} f(s) ds$, and $g(-1) = Ce^{-1}$. By substituting these values into Equation (3.2), we get

$$C - \int_{-1}^{0} e^{-s} f(s) ds - f(0) = B_0 \Big(C - \int_{-1}^{0} e^{-s} f(s) ds \Big) + B_1 C e^{-1},$$

and we can use it to find the constant C:

$$C - \int_{-1}^{0} e^{-s} f(s) ds - f(0) = B_0 C - B_0 \int_{-1}^{0} e^{-s} f(s) ds + B_1 C e^{-1},$$

by isolating C,

$$C = (I - B_0 - B_1 e^{-1})^{-1} \Big[(I - B_0) \int_{-1}^{0} e^{-s} f(s) ds + f(0) \Big]. \tag{3.3}$$

As C depends of the function $f(\theta)$, from now on we will call it $C(f)$.

Step 3: The last step is to apply $(I + A)$ to $((I - A)^{-1})(f(\theta))$,

$$\begin{aligned}
\varphi_{\mathcal{C}}(A)(f(\theta)) &= ((I + A)((I - A)^{-1}))(f(\theta)) \\
&= (I + A) \Big[C(f) e^{\theta} - \int_{-1}^{\theta} e^{\theta - s} f(s) ds \Big] \\
&= C(f) e^{\theta} - \int_{-1}^{\theta} e^{\theta - s} f(s) ds + C(f) e^{\theta} - \int_{-1}^{\theta} e^{\theta - s} f(s) ds - f(\theta) \\
&= -f(\theta) + 2C(f) e^{\theta} - 2 \int_{-1}^{\theta} e^{\theta - s} f(s) ds.
\end{aligned}$$

This concludes the proof. □

Remark 3.2.1. In the process is assumed that $(I-B_0-B_1e^{-1})$ is invertible, but what if it is not? If $(I - B_0 - B_1e^{-1})$ is not invertible, $\det((I - B_0 - B_1e^{-1})) = 0$, so by Definition 1.3.2, $\lambda = 1$ is a characteristic exponent, which means that the system is unstable. So, the assumptions made in the proof are valid.

To make future computations easier, we will apply a slight change to $\varphi_{\mathcal{C}}(A)$.

Definition 3.2.2. Let $f(\theta) \in \mathcal{C}([-1, 0])$. Then, the operator T is defined in three steps:

(i) Define f_1 by multiplying the operator $f(\theta)$ by e^{θ}: $f_1(\theta) = e^{\theta}f(\theta)$.

(ii) Define f_2 by applying $\varphi_{\mathcal{C}}(A)$ to $f_1(\theta)$:

$$\begin{aligned} f_2(\theta) = \varphi_{\mathcal{C}}(A)(e^{\theta}f(\theta)) &= -f_1(\theta) + 2C(e^{\theta}f(\theta))e^{\theta} - 2\int_{-1}^{\theta} e^{\theta-s}f_1(s)ds \\ &= -e^{\theta}f(\theta) + 2C(e^{\theta}f(\theta))e^{\theta} - 2\int_{-1}^{\theta} e^{\theta-s}e^{s}f(s)ds \\ &= -e^{\theta}f(\theta) + 2C(e^{\theta}f(\theta))e^{\theta} - 2\int_{-1}^{\theta} e^{\theta}f(s)ds. \end{aligned}$$

Computing the C that we got in Equation (3.3),

$$\begin{aligned} C(e^{\theta}f(\theta)) &= (I - B_0 - B_1e^{-1})^{-1}\Big[(I - B_0)\int_{-1}^{0} f(s)ds + f(0)\Big] \\ &= (I - B_0 - B_1e^{-1})^{-1}(I - B_0)\int_{-1}^{0} f(s)ds \\ &+ (I - B_0 - B_1e^{-1})^{-1}f(0). \end{aligned}$$

If we define $A = (I - B_0 - B_1e^{-1})^{-1}$ and $B = A(I - B_0)$,

$$C(e^{\theta}f(\theta)) = A\int_{-1}^{0} f(s)ds + Bf(0). \tag{3.4}$$

(iii) Define T by multiplying f_2 by $e^{-\theta}$:

$$\begin{aligned} Tf(\theta) &= e^{-\theta}(f_2(\theta)) \\ &= e^{-\theta}\Big(- e^{\theta}f(\theta) + 2C(e^{\theta}f(\theta))e^{\theta} - 2\int_{-1}^{\theta} e^{\theta}f(s)ds\Big) \\ &= -f(\theta) + 2C(e^{\theta}f(\theta)) - 2\int_{-1}^{\theta} f(s)ds \\ &= -f(\theta) + 2C(e^{\theta}f(\theta)) - 2\int_{-1}^{\theta} f(s)ds, \end{aligned}$$

and if we use the notation we just defined in Equation (3.4), we get

$$Tf(\theta) = -f(\theta) + 2A\int_{-1}^{0} f(s)ds + 2Bf(0) - 2\int_{-1}^{\theta} f(s)ds. \tag{3.5}$$

Remark 3.2.2. Another way to define the operator T, that we will use in the last section of this chapter, is by defining another operator:

$$(Kf)(\theta) := 2A\int_{-1}^{0} f(s)ds + 2Bf(0) - 2\int_{-1}^{\theta} f(s)ds, \tag{3.6}$$

so, $T = -I + K$.

3.3 Discretization of the Tustin operator

Unlike the semi-discretization method, now the operator remains fixed as we refine the discretization. To discretize T, first notice that $X := C([-1,0];\mathbb{C}^N) = C_P([-1,0];\mathbb{C}^N)\times\theta + 1/2$, where $\theta + 1/2$ is the space generated by the function $\theta \mapsto \theta + 1/2$.

We choose a sequence of subspaces $U_2 \subset \cdots U_{2M} \subset \cdots$ using as basis the functions

$$u_n(\theta) = e^{2\pi i n\theta}, \text{ for } n \in [-M+1, M-1], \quad \text{and} \quad u_M(\theta) = \theta + 1/2,$$

and dual basis is $u^n(\theta) = e^{2\pi i n\theta}$ ($\overline{u^n}(\theta) = e^{-2\pi i n\theta}$) and $u^M = \delta_0 - \delta_{-1}$ ($\overline{u^M}(\theta) = u^M(\theta)$) ; here, the functionals $g \in (\mathcal{C}([-1,0]))^*$ act as $g(f) := \int_{-1}^{0} f\overline{g}$. We can define ι_M and π_M like in Equations (2.6) and (2.7):

$$\iota_M(z_{-M+1,1},\ldots,z_{-M+1,N},\ldots,z_{M,1},\ldots,z_{M,N}) = \sum_{i,j} z_{i,j}(v_i \otimes \mathbf{e}_j), \tag{3.7}$$

$$\pi_M f = ((u^{-M+1}\otimes \mathbf{e}^1)f,\ldots,(u^M\otimes\mathbf{e}^N)f). \tag{3.8}$$

So, the operator T can be approximated by the matrix T^M with elements

$$T^M_{(n,i),(m,j)} = (u^n\otimes\mathbf{e}^i)(T(u_m\otimes\mathbf{e}_j)). \tag{3.9}$$

Before computing the matrix elements, we will see a short proposition that will help us to write some different elements as a Kronecker product.

Proposition 3.3.1. *Let $f(\theta)$ be a function defined in $\mathcal{C}([-1,0];\mathbb{C})$, $A \in \mathbb{C}^{N\times N}$ a constant matrix and $\mathbf{e}_j$ a vector of the basis of $\mathbb{C}^N$. Then,*

$$A(f(\theta)\otimes\mathbf{e}_j) = f(\theta)\otimes(A\mathbf{e}_j).$$

Proof. Using the definition of the Kronecker's product, and the basic properties of the matrix product, it is easy to see that

$$A(f(\theta) \otimes \mathbf{e}_j) = Af(\theta)\mathbf{e}_j = f(\theta)A\mathbf{e}_j = f(\theta) \otimes (A\mathbf{e}_j).$$

□

Also, we will compute beforehand some basic integrals to use during the process without further explanation:

(i)
- If $k \in \mathbb{Z} \setminus \{0\}$, then,

$$\int_{-1}^{0} e^{2\pi iks} ds = \frac{1}{2\pi ik}(1-1) = 0.$$

- If $k = 0$, then,

$$\int_{-1}^{0} 1\, ds = 1.$$

(ii)
- If $k \in \mathbb{Z} \setminus \{0\}$, then,

$$\int_{-1}^{\theta} e^{2\pi iks} ds = \frac{1}{2\pi ik}(e^{2\pi ik\theta} - 1).$$

- If $k = 0$, then,

$$\int_{-1}^{\theta} 1 ds = \theta + 1.$$

(iii)
- If $k \in \mathbb{Z} \setminus \{0\}$, then, by integrating by parts,

$$\int_{-1}^{0} (s+1)e^{2\pi iks} ds = \frac{1}{2\pi ik}.$$

- If $k = 0$, then,

$$\int_{-1}^{0} s + 1 ds = \frac{1}{2}.$$

(iv) For $k \in \mathbb{Z} \setminus \{0\}$,

$$\frac{1}{2\pi ik}\Big(\int_{-1}^{0} (e^{2\pi iks} - 1)e^{2\pi ils} ds\Big) = \frac{1}{2\pi ik}\Big(\int_{-1}^{0} (e^{2\pi i(k+l)s} - e^{2\pi ils}) ds\Big).$$

- If $k + l = 0$, by using integral (i),

$$\frac{1}{2\pi ik}\Big(\int_{-1}^{0} (1 - e^{2\pi ils}) ds\Big) = \frac{1}{2\pi ik}.$$

- If $k+l \neq 0$ and $l \neq 0$, by using integral (i),

$$\Big(\int_{-1}^{0} (e^{2\pi i(k+l)s} - e^{2\pi ils}) ds \Big) = 0.$$

- If $k+l \neq 0$ and $l = 0$, by using integral (i),

$$\frac{1}{2\pi ik} \Big(\int_{-1}^{0} (e^{2\pi iks} - 1) ds \Big) = -\frac{1}{2\pi ik}.$$

Now, we will proceed to find the elements of the matrix. First, we compute the action of T on the basis functions, using Equation (3.5):

$$\begin{aligned} T(u_m \otimes \mathbf{e}_j) = &-u_m(\theta) \otimes \mathbf{e}_j + 2A \int_{-1}^{0} (u_m(s) \otimes \mathbf{e}_j) ds \\ &+ 2B(u_m(0) \otimes \mathbf{e}_j) - 2 \int_{-1}^{\theta} (u_m(s) \otimes \mathbf{e}_j) ds. \end{aligned}$$

Now, we apply the dual basis functions to the previous equation:

$$\begin{aligned} &(u^n \otimes \mathbf{e}^i)(T(u_m \otimes \mathbf{e}_j)) \\ &\quad = (u^n \otimes \mathbf{e}^i) \Big(u_m(\theta) \otimes \mathbf{e}_j - 2K(u_m(\theta) \otimes \mathbf{e}_j) + 2 \int_{-1}^{\theta} u_m(s) \otimes \mathbf{e}_j ds \Big) \\ &\quad = (u^n \otimes \mathbf{e}^i)(u_m(\theta) \otimes \mathbf{e}_j) \\ &\qquad + (u^n \otimes \mathbf{e}^i)(-2K(u_m(\theta) \otimes \mathbf{e}_j)) \\ &\qquad + (u^n \otimes \mathbf{e}^i) \Big(2 \int_{-1}^{\theta} u_m(s) \otimes \mathbf{e}_j ds \Big). \end{aligned}$$

To make the computations easier, we will compute each term separately, analyzing the different cases that can appear for each term, and finally we will sum them up.

Notation 8. For the computations, we will use the next notation:

(i) $(T_1 f)(\theta) = -f(\theta)$,

(ii) $(T_2 f)(\theta) = 2A \int_{-1}^{0} f(s) ds$,

(iii) $(T_3 f)(\theta) = 2Bf(0)$,

(iv) $(T_4 f)(\theta) = -2 \int_{-1}^{\theta} f(s) ds$.

and $(Tf)(\theta) = (T_1 f)(\theta) + (T_2 f)(\theta) + (T_3 f)(\theta) + (T_4 f)(\theta)$. The same can be applied to the matrix elements, so

$$T^M_{(n,i),(m,j)} = T^M_{1,(n,i),(m,j)} + T^M_{2,(n,i),(m,j)} + T^M_{3,(n,i),(m,j)} + T^M_{4,(n,i),(m,j)} \quad (3.10)$$

With that notation, and using the integrals (i), (ii), (iii) and (iv), the computation of each

$$(u^n \otimes \mathbf{e}^i)(T_k(u_m \otimes \mathbf{e}_j)), \text{ for } k = 1,2,3,4, \forall n, m \in [-M+1, M], \forall i, j \in [0, N],$$

becomes straightforward. For each term, it is important to separate the cases where n or $m = -M+1, \dots, M-1$ and n or $m = M$.

If we sum up all the computations, we get that all the entries of the Tustin matrix are:

$$\begin{aligned}
T^M_{(0,i),(0,j)} &= -2\delta_{ij} + 2A_{ij} + 2B_{ij}, \\
T^M_{(0,i),(m,j)} &= 2A_{ij} + \frac{1}{\pi i m}\delta_{ij}, && \text{for } 0 < |m| < M \\
T^M_{(n,i),(0,j)} &= \frac{1}{\pi i n}\delta_{ij}, && \text{for } 0 < |n| < M \\
T^M_{(n,i),(n,j)} &= -\Big(1 + \frac{1}{\pi i n}\Big)\delta_{ij}, && \text{for } 0 < |n| < M \\
T^M_{(M,i),(0,j)} &= -2\delta_{ij}, \\
T^M_{(0,i),(M,j)} &= \frac{1}{6}\delta_{ij} + A_{ij}, \\
T^M_{(n,i),(M,j)} &= \frac{1}{2\pi i n}\Big(1 + \frac{1}{\pi i n}\Big)\delta_{ij}, \\
T^M_{(M,i),(M,j)} &= -\delta_{ij}.
\end{aligned}$$

After all this work, when doing the numerical tests presented in the next chapter, we will see that the Tustin method has severe deficiencies. We suspect that the reason behind it is that, with the space that we have chosen for the discretization, the Fourier series do not converge to the continuous functions that we are discretizing [8, Chapter 1.3].

To fix that problem, we will now present the Tustin second method (or the improved Tustin method).

3.4 The improved Tustin method

The main idea behind the improved Tustin method is to use a different space for the discretization, where the Fourier series converge to the continuous functions that we are discretizing. First, let us see some theory.

Lemma 3.4.1. *Let $H^k := H^k([-1,0];\mathbb{C}^N)$, for $k \geq 1$ integer. Then,*

(i) *The operator $T|_{H^k} : H^k \to H^k$ is a well-defined, bounded operator;*

(ii) *$T|_{H^k} = -I + K|_{H^k}$, where $K|_{H^k}$ is a compact operator; and*

(iii) *If $f \in C([-1,0];\mathbb{C}^N)$ is an eigenfunction of T, then $f \in H^k$, that is, $\sigma(T) = \sigma(T|_{H^k})$.*

With this lemma, we can choose the space $X := H^1([-1,0];\mathbb{C}^N)$. In this space, $g \in X^*$ acts as

$$g(f) = \langle f, g\rangle_1 := \int_{-1}^{0} (f \cdot \overline{g} + f' \cdot \overline{g}'). \tag{3.11}$$

In $H^1([-1,0])$, we define the orthonormal basis and its dual as:

$$\tilde{u}_n(\theta) = \tilde{u}^n(\theta) = \frac{e^{2\pi i n\theta}}{(1+4\pi^2 n^2)^{1/2}}, \quad \text{for } n \in [-M+1, M-1], \text{ and}$$
$$\tilde{u}_M(\theta) = \tilde{u}^M(\theta) = \frac{\sinh(\theta+1/2)}{\sqrt{\sinh 1}}.$$

To see that this basis is orthonormal, we just need to prove that, for $\forall n, m \in [-M+1, M]$,

$$\langle \tilde{u}_m(\theta), \tilde{u}^n(\theta)\rangle_1 = 0, \text{ if } m \neq n,$$
$$\langle \tilde{u}_m(\theta), \tilde{u}^n(\theta)\rangle_1 = 1, \text{ if } m = n,$$

which is straightforward using Equation (3.11).

The discretized operator T^M is defined as before, but now the matrix elements are computed using the basis $\tilde{u}_n \otimes \mathbf{e}^i$ and its dual $\tilde{u}^n \otimes \mathbf{e}_i$. The matrix elements are

$$\tilde{T}^M_{(n,i),(m,j)} = (\tilde{u}^n \otimes \mathbf{e}^i)(T(\tilde{u}_m \otimes \mathbf{e}_j)). \tag{3.12}$$

To justify the correctness of this method, in contraposition to the first Tustin method, we will see that the improved Tustin method converges in norm to T.

Theorem 3.4.2. $\|P_M K|_{H^1} P_M - K|_{H^1}\| \to 0$, *as* $M \to \infty$.

With this last result, we can use the results in [6] to justify that $\sigma(T^M)$ approaches $\sigma(T)$.

Both Lemma 3.4.1 and Theorem 3.4.2 are proved in the article [7].

So, with a solid theoretical background, the last step is to compute all the matrix elements of the improved Tustin method. The computations are similar to the ones made for the first Tustin method, but now with the new basis and dual basis. Repeating the process of Section 3.3, we get that all

the entries of the improved Tustin matrix are zero except for:

$$
\begin{aligned}
\tilde{T}^M_{(0,i),(0,j)} &= -2\delta_{ij} + 2B_{ij} + 2A_{ij}, \\
\tilde{T}^M_{(0,i),(m,j)} &= \frac{1}{\sqrt{1+4\pi^2m^2}}\Big(2A_{ij} + \frac{1}{\pi i m}\delta_{ij}\Big), && \text{for } 0 < |m| < M \\
\tilde{T}^M_{(n,i),(0,j)} &= \frac{1}{\pi i n\sqrt{1+4\pi^2n^2}}\delta_{ij}, && \text{for } 0 < |n| < M \\
\tilde{T}^M_{(n,i),(n,j)} &= -\Big(1 + \frac{1}{\pi i n}\Big)\delta_{ij}, && \text{for } 0 < |n| < M \\
\tilde{T}^M_{(M,i),(0,j)} &= -\frac{2\cosh(\frac{1}{2})}{\sqrt{\sinh(1)}}\delta_{ij}, \\
\tilde{T}^M_{(0,i),(M,j)} &= \frac{2\sinh(\frac{1}{2})A_{ij} + (2\cosh(\frac{1}{2}) - 4\sinh(\frac{1}{2}))\delta_{ij}}{\sqrt{\sinh(1)}}, \\
\tilde{T}^M_{(n,i),(M,j)} &= -\frac{4\sinh(\frac{1}{2})}{\sqrt{(1+4\pi^2n^2)\sinh(1)}}\delta_{ij}, \\
\tilde{T}^M_{(M,i),(M,j)} &= -\delta_{ij}
\end{aligned}
$$

Chapter 4

Numerical tests

To test the methods presented in the previous chapters, we will perform two different types of numerical tests. The first one will be a simple test, where we will apply the methods to the Hayes equation, which stability chart was presented in Chapter 1. The second test will be a more precise one, where we will compute the largest eigenvalues of some random matrices and verify how accurate those eigenvalues are. Apart from the precision of the eigenvalues, it is really important to compare the time of execution of the methods, as the main goal of this work is to find a method that is both precise and fast.

To find the largest eigenvalue, the fastest way is to use the Arnoldi's method, an iterative method that finds the largest eigenvalue of a matrix. The Arnoldi's method is a good choice for this work, as it is a fast method and it is able to find the largest eigenvalue of a matrix with a good precision. Still, we will se during this chapter that it presents some convergence errors. Arnoldi's method is implemented in Python in the library SciPy, in the function `scipy.sparse.linalg.eigs`.

4.1 Stability charts

Before going into a deeper analysis, a good way of seeing if a method works well is to construct the stability chart of a equation of which we already know the exact stability chart. In this case, we have chosen the Hayes equation, presented in Chapter 1, as it is one of the most basic examples of a linear non-periodic DDE with just one discrete time delay.

To compare the results, using Python, we can apply each method many times for different values of a and b and plot the results: if the system is stable in the point (a, b), we will plot a blue point, if it is unstable, we will plot a yellow point. Then, we can compare the results to the analytical stability chart of the Hayes equation shown in Figure 1.1. All the stability chart will be computed in the are $[-15, 15] \times [-15, 15]$.

Remark 4.1.1. It is important to remark that if a method is able to find the correct stability chart, that does not mean it is a good method. In the other hand, if a method is not able to find the correct stability chart, that means it is a bad method and we can discard it to further analysis.

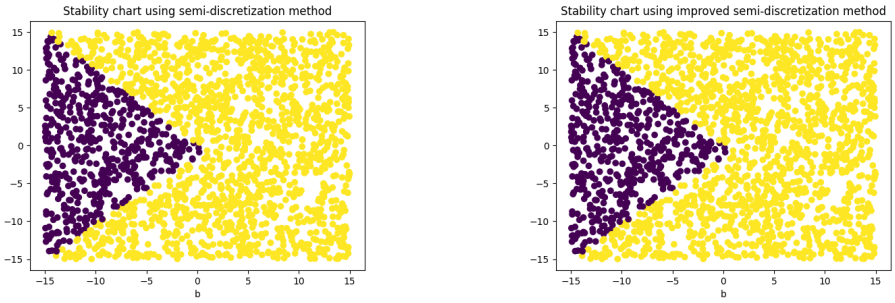

Figure 4.1: Stability chart of the Hayes equation using both Semi-discretization methods. The axis x represents the values of a and the axis y represents the values of b.

Both semi-discretization methods finds the correct stability chart of the Hayes equation, as we can see in the Figures 4.1.

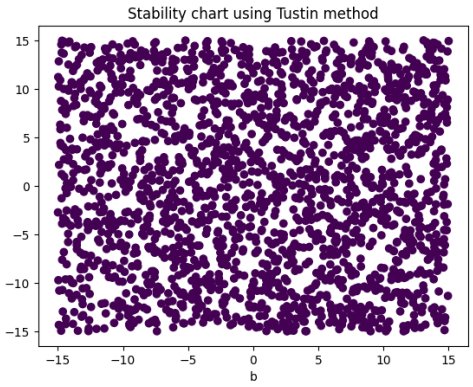

Figure 4.2: Stability chart of the Hayes equation using the Tustin method. The axis x represents the values of a and the axis y represents the values of b.

We can see that the method is not able to find the correct stability chart of the Hayes equation. This is a clear sign that the method is not good enough to be used in further analysis. A possible reason behind that is explained at the end of Chapter 3.

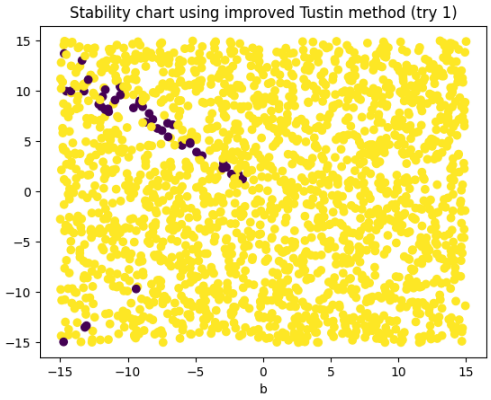

Figure 4.3: First try to build a stability chart using the Tustin method.

At first glance, the improved Tustin method does not seem able to find the correct stability chart. Still, this first seems more accurate than the original Tustin method.

Let us make a test to see where the improved Tustin method's eigenvalues are. To do so, an option is to set the size of the discretized matrix to 100×100 and use the NumPy function `numpy.linalg.eig` to find the exact eigenvalues, instead of using the Arnoldi's method.

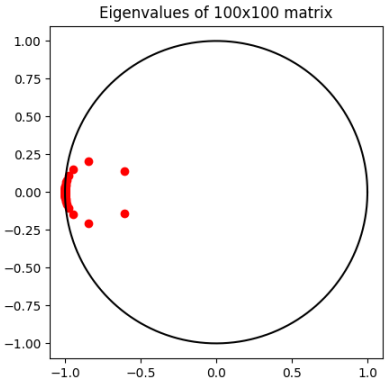

Figure 4.4: Exact eigenvalues of a 100×100 matrix in the imaginary plane, where the axis x represent the real part and the axis y represent the imaginary part.

We can clearly see that the majority of the eigenvalues are concentrated in the point $(-1, 0)$. If instead of taking a 100×100 matrix we take a $N \times N$ matrix, for a big N, the result would be the same: some few values inside

the circle and a lot of them around the point $(-1, 0)$. All those eigenvalues near the point $(-1, 0)$, if we apply the Cayley inverse, are points that are very large and with negative real part.

The reason behind that is that when finding the eigenvalues of a system, if we fix one eigenvalue, λ, there will always be infinite eigenvalues to the left of λ. So, when applying the Cayley transform, there will be infinite points around $(-1, 0)$. That presents a huge deficiency in the improved Tustin method, for three reasons:

(i) The Arnoldi's method used to find the largest eigenvalue often presents convergence errors near the point $(-1, 0)$.

(ii) If in the complex plane the right-most eigenvalue is, for example, $(-0.5, 0)$, that will not be the largest eigenvalue after applying the Cayley transform, as there are infinite eigenvalues around $(-1, 0)$. Even if all those eigenvalues still have negative real part, the system should be stable, but the way to conclude so is not as trivial at it should be.

(iii) In the eigenvalues around $(-1, 0)$, we have seen that, due to unknown numerical errors, some eigenvalues appear to be slightly outside the unit circle.

To solve the third problem, we can apply slight corrections to the code, so, if the largest eigenvalue is around the point $(-1, 0)$, but slightly outside the unit circle, we can manually define it as stable. This is a good solution, as the eigenvalues around $(-1, 0)$ are always very close to the unit circle, so we can define a small threshold to define if the eigenvalue is stable or not.

To solve the first problem, if there is no convergence, we can again manually define it as stable. We can make such change because we know that if there is one eigenvalue outside the unit circle and not near the point $(-1, 0)$, the Arnoldi's method will converge to it; so, if it does not converge, it is because the eigenvalue is near the point $(-1, 0)$, and we can define it as stable.

For the second problem, a possible solution given in [7] is to use a different transformation instead of $\varphi_{\mathcal{C}}$:

$$\varphi_\mu(z) = \frac{1 - \mu + z}{1 + \mu - z}.$$

Anyways, we will perform the tests without considering this problem.

With all those changes, let us see the stability chart of the improved Tustin method.

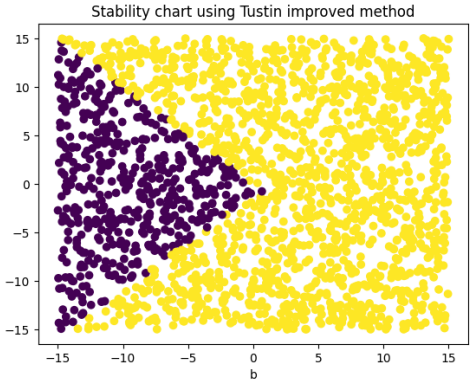

Figure 4.5: Stability chart of the Hayes equation using the improved Tustin method. The axis x represents the values of a and the axis y represents the values of b.

We can see that the method is now able to find the correct stability chart of the Hayes equation.

4.2 Precision of eigenvalues

To test the precision of the eigenvalues, we will compute the eigenvalues of some random matrices and compare the results. We will compare the results of the semi-discretization method, the improved semi-discretization method and the improved Tustin method, as we have already discarded the Tustin method in the previous section.

To do those comparisons, first we need to describe the method to find the error in an eigenvalue.

Lemma 4.2.1. *Let $M(\lambda) \in \mathbb{C}^{N\times N}$ be an $N \times N$ matrix. Then,*

$$\frac{d}{d\lambda}\det(M(\lambda)) = \operatorname{tr}\left(\operatorname{adj}(M(\lambda))\frac{d}{d\lambda}M(\lambda)\right)$$

This lemma is known as the Jacobi's formula, and its proof can be found in [7].

Lemma 4.2.2. *Let λ_{real} be a characteristic exponent and λ be an approximation of λ_{real}.*

$$\frac{|\lambda - \lambda_{real}|}{|\Delta\lambda|} \approx \frac{1}{|\Delta'(\lambda)|}.$$

Proof. From the definition of the derivative, and considering the fact that $\Delta(\lambda_{real}) = 0$, we have:

$$\frac{|\Delta\lambda|}{|\lambda - \lambda_{real}|} \approx |\Delta'(\lambda)|,$$

and the result follows. □

Theorem 4.2.3. *Let λ_{real} be a characteristic exponent and λ be an approximation of λ_{real}. The error $|\lambda - \lambda_{real}|$ can be approximated as:*

$$|\lambda - \lambda_{real}| \approx \frac{|\Delta(\lambda)|}{|\operatorname{tr}(\operatorname{adj}((\lambda I_N - B_0 - e^{-\lambda}B_1)(I_N + e^{-\lambda}B_1)))|}.$$

Proof.

$$|\lambda - \lambda_{real}| = |\lambda - \lambda_{real}|\frac{|\Delta(\lambda)|}{|\Delta(\lambda)|} = \frac{|\lambda - \lambda_{real}|}{|\Delta(\lambda)|}|\Delta(\lambda)|.$$

Now, by using Lemma 4.2.2, we have the approximation $\frac{|\lambda-\lambda_{real}|}{|\Delta(\lambda)|} \approx \frac{1}{|\Delta'(\lambda)|}$, so

$$|\lambda - \lambda_{real}| \approx \frac{|\Delta(\lambda)|}{|\Delta'(\lambda)|}.$$

Using Lemma 4.2.1, we can write $\Delta'(\lambda)$ as
$\operatorname{tr}(\operatorname{adj}((\lambda I_N - B_0 - e^{-\lambda}B_1)(I_N + e^{-\lambda}B_1)))$. Hence,

$$|\lambda - \lambda_{real}| \approx \frac{|\Delta(\lambda)|}{|\operatorname{tr}(\operatorname{adj}((\lambda I_N - B_0 - e^{-\lambda}B_1)(I_N + e^{-\lambda}B_1)))|}.$$

□

Remark 4.2.1. It is important to remark that from the eigenvalues of S^h, $\tilde{S}^h$ and $\tilde{T}^M$, we must undo the exponential function and the Cayley transform in order to find the approximation of the characteristic exponents. Without recovering the original eigenvalues, the error would be meaningless.

Proposition 4.2.4. *Let μ be an eigenvalue of S^h. Then, the approximation of the characteristic exponent, λ, is given by the inverse of the exponential function:*

$$\lambda = \frac{1}{h}\log(\mu),$$

where $\log(\mu)$ is the complex logarithm of μ. The same is valid for the eigenvalues of $\tilde{S}^h$.

It is important to remark that the exponential function is not injective. Still, for a sufficiently small h, this is not a problem, as the imaginary part of the eigenvalues of A is small and lies in $\{z : |\operatorname{Im} z| < \frac{\pi}{h}\}$ [7].

Proof. The proof is trivial, using the relation stated in theorem 2.3.1. □

Proposition 4.2.5. *Let μ be an eigenvalue of $\tilde{T}^M$. Then, the approximation of the characteristic exponent, λ, is given by the inverse of the Cayley transform:*

$$\lambda = \frac{\mu + 1}{\mu - 1}.$$

Proof. The proof is trivial, as the Cayley transform is a type of Möbius transform. □

Remark 4.2.2. As we are studying the stability of a system, we only need to verify the precision in finding the largest eigenvalue. If we want more than one eigenvalue, more numerical tests must be performed. Those tests are made in [7].

With all that, we just need to compare the precision in finding the largest eigenvalue for each method. To do so, we will graph the error in finding the largest eigenvalue for each method, for random matrices B_0 and B_1 with size $N = 3$, for increasing dimensions.

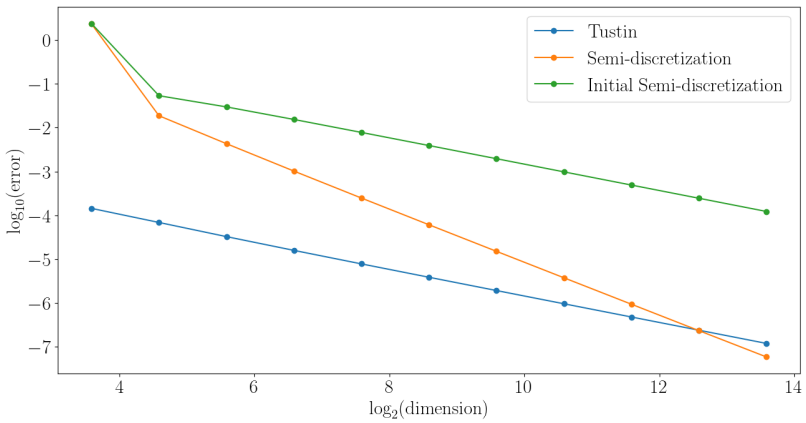

Figure 4.6: Comparison between the error in finding the largest eigenvalue. The axis x represents the $\log_2$ of the order of approximation (or the dimension) and the axis y represents the $\log_{10}$ of the error in finding the largest eigenvalue.

In terms of precision, the Semi-discretization method is the worst, as it presents presents the biggest error for every value of order of approximation. Comparing improved Tustin method and improved Semi-discretization method, the improved Tustin method is better for small orders of approximation, but for the biggest one the improved Semi-discretization method is slightly better, what means that the improved Semi-discretization method converges better to the largest eigenvalue.

That comparison was made taking into account only the precision of the largest eigenvalue. If we take into account the time of execution, we can draw some better conclusions.

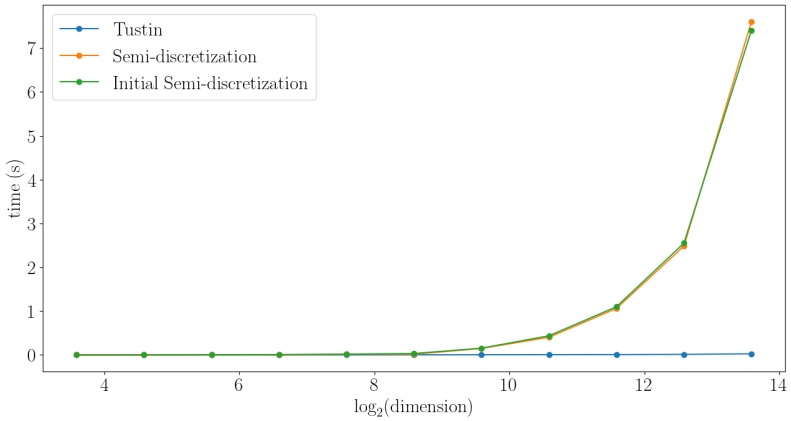

Figure 4.7: Comparison between the time of execution of finding the largest eigenvalue. The axis x represents the $\log_2$ of the order of approximation (or the dimension) and the axis y represents the time of execution.

We can clearly see the difference in time of execution between the methods. Even if the improved Semi-discretization method converges better in terms of precision, the amount of time needed is much bigger than the improved Tustin method. Because of the sparsity of the matrices in the construction of the improved Tustin method, the time of execution is really good, as it can easily handle big matrices.

To compare how better the improved Tustin method is in terms of time of execution, we can repeat Figures 4.6 and 4.7 for bigger orders of approximation just for the improved Tustin method, until its time of execution is similar to the ones of the other methods.

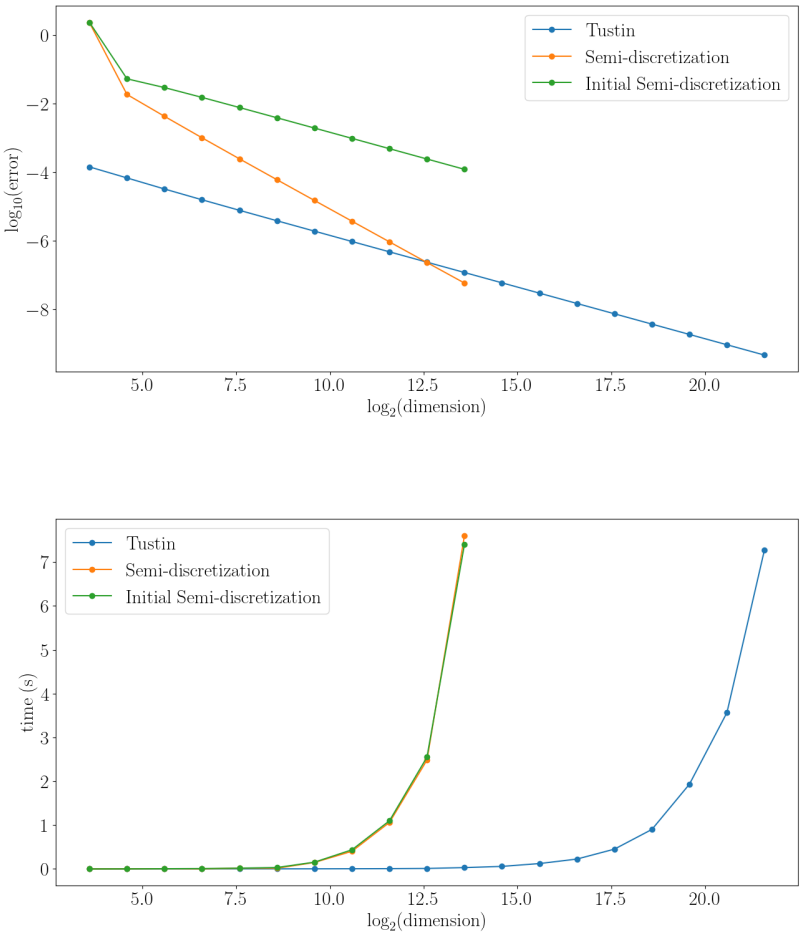

Figure 4.8: Comparison between the methods of error in finding the largest eigenvalue (up), and the execution time (down).

We can see that for the same time of execution, the improved Tustin method gives better results.

Chapter 5

Conlcusions

After the numerical tests, we can draw some general conclusions.

First, the Tustin method is not a suitable method for the analysis of time-delay systems. In comparison, the well developed and widely used Semi-discretization method, seems to be far worse than the remaining two method, as its improved version has a similar time of execution and much better accuracy, together with a better theoretical background.

Now, comparing the two remaining method, the improved Tustin method is much better in term of execution time, because of its sparsity. Also, it seems to be more accurate than the Semi-discretization method. Still, for really big dimensions, the improved Semi-discretization method is better in terms of accuracy, as it converges better than the improved Tustin method; but, to acquire this accuracy, the execution time could be too big to handle.

A big disadvantage of the improved Tustin method is the fact that it has an accumulation point in $\{-1\}$, which could lead to numerical instability. In this work, that numerical stability was solved with some manual corrections, but in a real application, this has to be fixed and could present a big problem.

In conclusion, for the systems analyzed in this work, the improved Tustin method seems the best choice, as it has a good balance between execution time and accuracy. Still, it is not a well established method as the Semi-discretization method, and it could present some problems in a real application. Also, the improved Semi-discretization method seems to be a good improvement over the original Semi-discretization method, and it does not have the problems of the improved Tustin method, even if it is much slower.

Appendix A

Python code

A.1 Building matrices

```
# ==== Imports ====
from itertools import product

import numpy as np
from scipy.linalg import inv, expm
from scipy.sparse import csr_matrix

# ==== Functions ====
inv_cayley = lambda x : (x - 1) / (x + 1)

def cofactor_matrix(M):
    M = np.asarray(M)
    size = M.shape[0]
    C = np.zeros_like(M)
    # remove row i and column j from M.
    for i, j in product(range(size), repeat=2):
        C[i, j] = (-1)**(i+j)*np.linalg.det(M[np.arange(size)!=
    i, :][:, np.arange(size)!=j])
    return C

def jacobi(M, dM):
    return np.trace(cofactor_matrix(M).T @ dM)

def test_eigenvalue(B0, B1, eigval):
    if np.isnan(eigval):
        return np.nan
    B0 = np.asarray(B0)
    B1 = np.asarray(B1)
    N = B0.shape[0]
    Id = np.identity(N)
    L = eigval * Id - B0 - np.exp(-eigval) * B1
    dL = Id + np.exp(-eigval) * B1
    d_dis = jacobi(L, dL)
    return np.abs(np.linalg.det(L) / d_dis)

```

```
35 def _transfer_matrix(M, data, row_ind,
36                       col_ind, m1:int, m2:int):
37     N1, N2 = M.shape
38     M = [(M[i, j], m1 + i, m2 + j) for i, j in product(range(N1
       ), range(N2))]
39     local_data, local_row, local_col = list(zip(*M))
40     data += local_data
41     row_ind += local_row
42     col_ind += local_col
43
44 # ==== Semi-discretization matrices ====
45 def initial_semi_discretization_matrix(B_0, B_1, M:int, tau=1):
46     # correct effect of time delay.
47     B_0 = np.copy(tau * B_0)
48     B_1 = np.copy(tau * B_1)
49     h = 1 / (M-1)
50     # basic definitions.
51     N = np.shape(B_0)[0]
52     Id = np.identity(N)
53     A = inv(B_0)
54     B = expm(h * B_0)
55     # build the matrix.
56     data = []
57     row_ind, col_ind = [], []
58     # (M - 1, 0) block
59     S = ((B - Id) @ A @ B_1)
60     _transfer_matrix(
61         S, data, row_ind, col_ind, (M - 1) * N, 0)
62     # (M - 1, M - 1) block
63     S = B
64     _transfer_matrix(
65         S, data, row_ind, col_ind, (M - 1) * N, (M - 1) * N)
66     # subdiagonal elements.
67     for m in range(1, M):
68         S = Id
69         _transfer_matrix(
70             S, data, row_ind, col_ind, (m-1) * N, m * N)
71
72     return csr_matrix((data, (row_ind, col_ind)), shape=(M*N, M
       *N))
73
74 def semi_discretization_matrix(B_0, B_1, M:int, tau=1):
75     # correct effect of time delay.
76     B_0 = np.copy(tau * B_0)
77     B_1 = np.copy(tau * B_1)
78     h = 1 / (M-1)
79     # basic definitions.
80     N = np.shape(B_0)[0]
81     Id = np.identity(N)
82     A = inv(B_0)
83     B = expm(h * B_0)
84     # build the matrix.
85     data = []
86     row_ind, col_ind = [], []
```

```
87      # (M - 1, 0) block
88      S = A @ (B + (1/h)*A - (1/h)*A @ B) @ B_1
89      _transfer_matrix(
90          S, data, row_ind, col_ind, (M - 1) * N, 0)
91      # (M - 1, 1) block
92      S = -A @ (Id + (1/h)*A - (1/h)*A @ B) @ B_1
93      _transfer_matrix(
94          S, data, row_ind, col_ind, (M - 1) * N, N)
95      # (M - 1, M - 1) block
96      S = B
97      _transfer_matrix(
98          S, data, row_ind, col_ind, (M - 1) * N,
99          (M - 1) * N)
100
101     # subdiagonal elements.
102     for m in range(1, M):
103         S = Id
104         _transfer_matrix(
105             S, data, row_ind, col_ind, (m-1) * N, m * N)
106
107     return csr_matrix((data, (row_ind, col_ind)), shape=(M*N, M
        *N))
108
109 # ==== Tustin matrices ====
110 def tustin_matrix_unstable(B_0, B_1, M:int, tau=1):
111     # correct effect of time delay.
112     B_0 = np.copy(tau * B_0)
113     B_1 = np.copy(tau * B_1)
114     # basic definitions.
115     N = np.shape(B_0)[0]
116     Id = np.identity(N, dtype=complex)
117     A = 2 * inv(Id - B_0 - np.exp(-1) * B_1)
118     B = A @ (Id - B_0)
119
120     # build the matrix.
121     data = []
122     row_ind, col_ind = [], []
123     # (0, 0) block
124     T = -2 * Id + A + B
125     _transfer_matrix(
126         T, data, row_ind, col_ind, 0, 0)
127     # (M, 0) block
128     T = -2 * Id
129     _transfer_matrix(
130         T, data, row_ind, col_ind, (2*M-1)*N, 0)
131     # (0, M) block
132     T = (1/6) * Id + 0.5 * A
133     _transfer_matrix(
134         T, data, row_ind, col_ind, 0, (2*M-1)*N)
135     # (M, M) block
136     T = -Id
137     _transfer_matrix(
138         T, data, row_ind, col_ind, (2*M-1)*N, (2*M-1)*N)
139
```

```
140     for n in range(1, M):
141         # even indices are for positive values of n.
142         # diagonal elements.
143         T = -(1 + (1 / (np.pi * 1j * n))) * Id
144         _transfer_matrix(
145             T, data, row_ind, col_ind, (2*n)*N, (2*n)*N)
146         # 0th row
147         T = A + (1 / (np.pi * 1j * n)) * Id
148         _transfer_matrix(
149             T, data, row_ind, col_ind, 0, (2*n)*N)
150         # 0th column
151         T = (1 / (np.pi * 1j * n)) * Id
152         _transfer_matrix(
153             T, data, row_ind, col_ind, (2*n)*N, 0)
154         # Mth column
155         T = (1 / (2 * np.pi * 1j * n)) * (1 + 1 / (np.pi * 1j *
    n)) * Id
156         _transfer_matrix(
157             T, data, row_ind, col_ind, (2*n)*N, (2*M-1)*N)
158 
159         # odd indices are for negative values of n.
160         # diagonal elements.
161         T = -(1 - (1 / (np.pi * 1j * n))) * Id
162         _transfer_matrix(
163             T, data, row_ind, col_ind, (2*n-1)*N, (2*n-1)*N)
164         # 0th row
165         T = A - (1 / (np.pi * 1j * n)) * Id
166         _transfer_matrix(
167             T, data, row_ind, col_ind, 0, (2*n-1)*N)
168         # 0th column
169         T = -(1 / (np.pi * 1j * n)) * Id
170         _transfer_matrix(
171             T, data, row_ind, col_ind, (2*n-1)*N, 0)
172         # Mth column
173         T = (1 / (2 * np.pi * 1j * n)) * (-1 + 1 / (np.pi * 1j
    * n)) * Id
174         _transfer_matrix(
175             T, data, row_ind, col_ind, (2*n-1)*N, (2*M-1)*N)
176 
177     return csr_matrix((data, (row_ind, col_ind)),
178     shape=(2*M*N, 2*M*N))
179 
180 def tustin_matrix_stable(B_0, B_1, M:int, tau=1):
181     # correct effect of time delay.
182     B_0 = np.copy(tau * B_0)
183     B_1 = np.copy(tau * B_1)
184     # basic definitions.
185     N = np.shape(B_0)[0]
186     Id = np.identity(N, dtype=complex)
187     A = 2 * inv(Id - B_0 - np.exp(-1) * B_1)
188     B = A @ (Id - B_0)
189 
190     # build the matrix.
191     data = []
```

```
    row_ind, col_ind = [], []
    # (0, 0) block
    T = -2 * Id + A + B
    _transfer_matrix(
        T, data, row_ind, col_ind, 0, 0)
    # (M, 0) block
    T = -2 * Id * np.cosh(1/2) / np.sqrt(np.sinh(1))
    _transfer_matrix(
        T, data, row_ind, col_ind, (2*M-1)*N, 0)
    # (0, M) block
    tmp = 2 * (np.cosh(1/2) - 2 * np.sinh(1/2)) * Id + (A * np.
sinh(1/2))
    T = tmp / np.sqrt(np.sinh(1))
    _transfer_matrix(
        T, data, row_ind, col_ind, 0, (2*M-1)*N)
    # (M, M) block
    T = -Id
    _transfer_matrix(
        T, data, row_ind, col_ind, (2*M-1)*N, (2*M-1)*N)

    for n in range(1, M):
        den = np.sqrt(1 + (2 * np.pi * n)**2)
        # even indices are for positive values of n.
        # diagonal elements.
        T = -(1 + (1 / (np.pi * 1j * n))) * Id
        _transfer_matrix(
            T, data, row_ind, col_ind, (2*n)*N, (2*n)*N)
        # 0th row
        T = (A + Id / (np.pi * 1j * n)) / den
        _transfer_matrix(
            T, data, row_ind, col_ind, 0, (2*n)*N)
        # 0th column
        T = Id / (np.pi *1j * n * den)
        _transfer_matrix(
            T, data, row_ind, col_ind, (2*n)*N, 0)
        # Mth column
        T = -4 * np.sinh(1/2) * Id / (den * np.sqrt(np.sinh(1))
)
        _transfer_matrix(
            T, data, row_ind, col_ind, (2*n)*N, (2*M-1)*N)

        # odd indices are for negative values of n.
        # diagonal elements.
        T = -(1 - (1 / (np.pi * 1j * n))) * Id
        _transfer_matrix(
            T, data, row_ind, col_ind, (2*n-1)*N, (2*n-1)*N)
        # 0th row
        T = (A - Id / (np.pi * 1j * n)) / den
        _transfer_matrix(
            T, data, row_ind, col_ind, 0, (2*n-1)*N)
        # 0th column
        T = -Id / (np.pi *1j * n * den)
        _transfer_matrix(
            T, data, row_ind, col_ind, (2*n-1)*N, 0)
```

```
        # Mth column
        T = -4 * np.sinh(1/2) * Id / (den * np.sqrt(np.sinh(1))
    )
        _transfer_matrix(
            T, data, row_ind, col_ind, (2*n-1)*N, (2*M-1)*N)

    return csr_matrix((data, (row_ind, col_ind)), shape=(2*M*N,
     2*M*N))
```

A.2 Stability charts

```
# ==== Imports ====
import math
import matplotlib.pyplot as plt
import numpy as np
import scipy
import itertools
from scipy.integrate import quad
from numpy.linalg import eigvals, eig
from scipy.linalg import expm
from scipy.linalg import inv
from scipy.sparse.linalg import eigs, ArpackNoConvergence
from itertools import product
from scipy import sparse
from scipy.sparse import csr_matrix
# ==== Semi-discretization ====
def test_stability_semi_discretization(b0, b1, M):
    b0 = np.asarray(b0)
    b1 = np.asarray(b1)
    approx = semi_discretization_matrix(b0, b1, M)
    v0 = np.ones(M, dtype=complex)
    try:
        mu = eigs(
            approx,
            k = 1,
            v0 = v0,
            which = 'LM',
            return_eigenvectors = False
        )[0]
    except ArpackNoConvergence:
        print('Convergence failed.')

    mu = np.log(mu)*(M-1)
    return np.real(mu)

rng = np.random.default_rng(seed=64343)
bs = rng.uniform(low=-15, high=15, size=(2000, 2, 1))
M = 20
data = [test_stability_semi_discretization([b[0]], [b[1]], M)
    for b in bs]
data = np.stack(data)

```

```
fig, axs = plt.subplots()
# remove last axis from bs.
bs = bs[:, :, 0]
# set value of data to 1 is > 0.
color = np.zeros_like(data)
color[data > 0] = 1
axs.scatter(*bs.T, c=color)

axs.set_xlabel('a')
axs.set_xlabel('b')
axs.set_title("Stability chart using improved semi-
    discretization method")

plt.show()
# ==== Improved Semi-discretization ====
def test_stability_initial_semi_discretization(b0, b1, M):
    b0 = np.asarray(b0)
    b1 = np.asarray(b1)
    approx = initial_semi_discretization_matrix(b0, b1, M)
    v0 = np.ones(M, dtype=complex)
    try:
        mu = eigs(
            approx,
            k = 1,
            v0 = v0,
            which = 'LM',
            return_eigenvectors = False
        )[0]
    except ArpackNoConvergence:
        print('Convergence failed.')
    mu = np.log(mu)*(M-1)
    return np.real(mu)

rng = np.random.default_rng(seed=64343)
bs = rng.uniform(low=-15, high=15, size=(2000, 2, 1))
M = 20
data = [test_stability_initial_semi_discretization([b[0]], [b
    [1]], M) for b in bs]
data = np.stack(data)


fig, axs = plt.subplots()
# re    move last axis from bs.
bs = bs[:, :, 0]
# set value of data to 1 is > 0.
color = np.zeros_like(data)
color[data > 0] = 1
axs.scatter(*bs.T, c=color)

axs.set_xlabel('a')
axs.set_xlabel('b')
axs.set_title("Stability chart using semi-discretization method
    ")

```

```
92 plt.show()
93 # ==== Tustin ====
94 def test_stability_tustin_unstable(b0, b1, M):
95     approx = tustin_matrix_unstable([b0], [b1], M)
96     v0 = np.zeros(2*M, dtype=complex)
97     v0[0] = 1
98     try:
99         mu = eigs(
100             approx,
101             k = 1,
102             v0 = v0,
103             which = 'LM',
104             return_eigenvectors = False
105         )[0]
106     except ArpackNoConvergence:
107         mu = 1
108 
109     mu = matrices.inv_cayley(mu)
110     return np.real(mu)
111 
112 rng = np.random.default_rng(seed=64343)
113 bs = rng.uniform(low=-15, high=15, size=(2000, 2, 1))
114 M = 20
115 data = [test_stability_tustin_unstable(b[0], b[1], M) for b in
        bs]
116 data = np.stack(data)
117 
118 
119 fig, axs = plt.subplots()
120 # re    move last axis from bs.
121 bs = bs[:, :, 0]
122 # set value of data to 1 is > 0.
123 color = np.zeros_like(data)
124 color[data > 0] = 1
125 axs.scatter(*bs.T, c=color)
126 
127 axs.set_xlabel('a')
128 axs.set_xlabel('b')
129 axs.set_title("Stability chart using Tustin method")
130 
131 plt.show()
132 # ==== Improved Tustin (try 1) ====
133 def test_stability_tustin_stable(b0, b1, M):
134     approx = matrices.tustin_matrix_stable([b0], [b1], M)
135     v0 = np.zeros(2*M, dtype=complex)
136     v0[0] = 1
137     try:
138         mu = eigs(
139             approx,
140             k = 1,
141             v0 = v0,
142             which = 'LM',
143             return_eigenvectors = False
144         )[0]
```

```
145     except ArpackNoConvergence:
146         mu = 1
147 
148     mu = matrices.inv_cayley(mu)
149     return np.real(mu)
150 
151 rng = np.random.default_rng(seed=64343)
152 bs = rng.uniform(low=-15, high=15, size=(2000, 2, 1))
153 M = 20
154 data = [test_stability_tustin_stable(b[0], b[1], M) for b in bs
        ]
155 data = np.stack(data)
156 
157 fig, axs = plt.subplots()
158 # re    move last axis from bs.
159 bs = bs[:, :, 0]
160 # set value of data to 1 is > 0.
161 color = np.zeros_like(data)
162 color[data > 0] = 1
163 axs.scatter(*bs.T, c=color)
164 
165 axs.set_xlabel('a')
166 axs.set_xlabel('b')
167 axs.set_title("Stability chart using improved Tustin method (
        try 1)")
168 
169 plt.show()
170 # ==== Eigenvalues circle ====
171 landa = lambda nu : (nu - 1) / (nu + 1)
172 B0 = [-5]
173 B1 = [0]
174 M = 50
175 # matrix = tustin_matrix_build_2(B0, B1, M)
176 matrix = matrices.tustin_matrix_stable(B0, B1, M).todense()
177 mu = eig(matrix)[0]
178 # diff = lambda_difference(B0, B1, mu)
179 
180 fig, ax = plt.subplots()
181 # draw a unit circle.
182 theta = np.linspace(0, 2*np.pi, 1000)
183 ax.plot(np.cos(theta), np.sin(theta), color='black')
184 # draw the eigenvalues.
185 # orig_eigen = matrices.inv_cayley(mu)
186 ax.scatter(mu.real, mu.imag, color='red')
187 # aspect ratio 1.
188 ax.set_aspect('equal', 'box')
189 
190 ax.set_title("Eigenvalues of 100x100 matrix")
191 plt.show()
192 # ==== Improved Tustin ====
193 def test_stability_tustin_stable(b0, b1, M):
194     if np.abs(1 - b0 - np.exp(-1) * b1) < 1e-2:
195         return 1
196     approx = matrices.tustin_matrix_stable([b0], [b1], M)
```

```
    approx = 1e-3 * np.identity(approx.shape[0]) + approx
    v0 = np.zeros(2*M, dtype=complex)
    v0[0] = 1
    try:
        mu = eigs(
            approx,
            k = 1,
            v0 = v0,
            which = 'LM',
            return_eigenvectors = False
        )[0]
        mu = mu - 1e-3
        if (np.abs(np.abs(mu) - 1) < 1e-2) and \ np.abs(np.angle(mu)) > 3 * np.pi / 4:
            mu = 1
        else:
            mu = 1 if np.abs(mu) < 1 else mu
    except ArpackNoConvergence:
        mu = 1

    mu = matrices.inv_cayley(mu)
    return np.real(mu)

rng = np.random.default_rng(seed=64343)
bs = rng.uniform(low=-15, high=15, size=(2000, 2, 1))
M = 20
data = [test_stability_tustin_stable(b[0], b[1], M) for b in bs]
data = np.stack(data)

fig, axs = plt.subplots()
# re    move last axis from bs.
bs = bs[:, :, 0]
# set value of data to 1 is > 0.
color = np.zeros_like(data)
color[data > 0] = 1
axs.scatter(*bs.T, c=color)

axs.set_xlabel('a')
axs.set_xlabel('b')
axs.set_title("Stability chart using Tustin improved method")

plt.show()
```

A.3 Errors and times

```
# ==== Imports ====
# Add path to module for stability
import os
import time
import sys
```

```
sys.path.insert(0, 'c:/Users/marke/OneDrive/Escritorio/TFG/tfg/src')

import numpy as np
import pandas as pd
from scipy.sparse.linalg import eigs, ArpackNoConvergence
from scipy.stats import linregress
import matplotlib.pyplot as plt
n_rep, app_low, app_high = 10, 1, 12
# app_high = 20 in Improved Tustin for similar time
# ==== Improved Tustin precision ====
# == Save results ==
# Check if there exists directory 'cache'.
if not os.path.exists('cache'):
    os.makedirs('cache')

# Check if a file with stored results exists.
fname = os.path.join('cache', 'tustin_stable_precision.csv')
if os.path.exists(fname):
    df = pd.read_csv(fname)
else:
    # Generate random positive matrices.
    rng = np.random.default_rng(seed=842463)
    N = 3
    n_eigs = 1
    B0 = rng.normal(scale=10, size=(N, N))
    B1 = rng.normal(scale=10, size=(N, N))
    # Set dimensions of the Tustin matrices.
    approx = 2**np.arange(app_low, app_high)
    errors = []
    mus = []
    final_times = []
    for M in approx:
        print(f'Approximation: {M}')
        matrix = tustin_matrix_stable(B0, B1, M)
        # Set function 1 as initial guess.
        v0 = np.zeros(2*M*N, dtype=complex)
        v0[:N] = 1
        times = []
        for _ in range(n_rep):
            start = time.time()
            mu = eigs(
                matrix,
                k = n_eigs,
                v0 = v0,
                which = 'LM',
                return_eigenvectors = False
            )
            end = time.time()
            times.append(end - start)
        if min(times) != 0:
            final_times.append(min(times))
        else:
            final_times.append(sum(times) / n_rep)
```

```
59          mu = matrices.inv_cayley(mu)
60          error = [test_eigenvalue(B0, B1, e)
61                   for e in mu]
62          error = np.array(error)
63          # Sort error by real part.
64          idx = np.argsort(np.real(mu))
65          error = error[idx]
66          mu = mu[idx]
67          errors.append(error)
68          mus.append(mu)
69      mus = np.stack(mus)
70      errors = np.stack(errors)
71      final_times = np.array(final_times)
72
73      # Create df to store the results.
74      df = pd.DataFrame()
75      df['dim'] = 2 * approx * N
76      df['time (s)'] = final_times
77      # Add eigenvalues and errors to the df.
78      for i in range(len(mu)):
79          df[f'eig_{i}'] = mus[:, i]
80      for i in range(len(mu)):
81          df[f'error_{i}'] = errors[:, i]
82      # Save df to a csv file.
83      df.to_csv(fname, index=False, sep=',')
84
85  # == Plot results ==
86  # Set format for plots.
87  plt.rcParams.update(plt.rcParamsDefault)
88  plt.rcParams['text.usetex'] = True
89  plt.rcParams['font.size'] = 22
90  plt.rcParams['font.family'] = 'serif'
91
92  # Set colors to identify the eigenvalues.
93  colors = ['maroon']
94
95  fig, axs = plt.subplots()
96  axs.plot(
97      np.log2(df['dim']), np.log10(df[f'error_{0}']), 'o-', label
        =f'eig {0}', color=colors[0])
98  axs.set_xlabel(r'$\log_2$(order of approximation)')
99  axs.set_ylabel(r'$\log_{10}$(error)')
100 axs.legend(loc='center left', bbox_to_anchor=(1, 0.5))
101
102 r = linregress(np.log2(df['dim']), np.log10(df['error_0']))
103 print(r)
104 print(f'Exponent: {r.slope / np.log10(2)}')
105
106 fig.set_size_inches(10, 8)
107 plt.show()
108 # ==== Semi-discretization precision ====
109 # == Save results ==
110 # Check if there exists directory 'cache'.
111 if not os.path.exists('cache'):
```

```
    os.makedirs('cache')

# Check if a file with stored results exists.
fname = os.path.join('cache', 'initial_semi_precision.csv')
if os.path.exists(fname):
    df = pd.read_csv(fname)
else:
    # Generate random positive matrices.
    rng = np.random.default_rng(seed=842463)
    N = 3
    B0 = rng.normal(scale=10, size=(N, N))
    B1 = rng.normal(scale=10, size=(N, N))
    # Set dimensions of the Tustin matrices.
    approx = 2 * 2**np.arange(app_low, app_high)
    errors = []
    mus = []
    final_times = []
    for M in approx:
        print(f'Approximation: {M}')
        matrix = initial_semi_discretization_matrix(B0, B1, M)
        # Set function 1 as initial guess.
        v0 = np.ones(M * N)
        times = []
        for _ in range(n_rep):
            start = time.time()
            mu = eigs(
                matrix,
                k = 1,
                v0 = v0,
                which = 'LM',
                return_eigenvectors = False
            )
            end = time.time()
            times.append(end - start)
        if min(times) != 0:
            final_times.append(min(times))
        else:
            final_times.append(sum(times) / n_rep)
        mu = np.log(mu) * (M - 1)
        mu = np.array(mu)
        error = [test_eigenvalue(B0, B1, e) for e in mu]
        error = np.array(error)
        errors.append(error)
        mus.append(mu)
    mus = np.stack(mus)
    errors = np.stack(errors)
    final_times = np.array(final_times)

    # Create df to store the results.
    df = pd.DataFrame()
    df['dim'] = approx * N
    df['time (s)'] = final_times
    # Add eigenvalues and errors to the df.
    for i in range(mus.shape[1]):
```

```
166         df[f'eig_{i}'] = mus[:, i]
167     for i in range(mus.shape[1]):
168         df[f'error_{i}'] = errors[:, i]
169     # Save df to a csv file.
170     df.to_csv(fname, index=False, sep=',')
171
172
173 # == Plot results ==
174 # Set format for plots.
175 plt.rcParams.update(plt.rcParamsDefault)
176 plt.rcParams['text.usetex'] = True
177 plt.rcParams['font.size'] = 22
178 plt.rcParams['font.family'] = 'serif'
179
180 # Set colors to identify the eigenvalues.
181 colors = ['maroon']
182
183 fig, axs = plt.subplots()
184 axs.plot(
185     np.log2(df['dim']), np.log10(df[f'error_{0}']), 'o-', label
    =f'eig {0}', color=colors[0])
186 axs.set_xlabel(r'$\log_2$(order of approximation)')
187 axs.set_ylabel(r'$\log_{10}$(error)')
188 axs.legend(loc='center left', bbox_to_anchor=(1, 0.5))
189
190 r = linregress(np.log2(df['dim']), np.log10(df['error_0']))
191 print(r)
192 print(f'Exponent: {r.slope / np.log10(2)}')
193
194 fig.set_size_inches(10, 8)
195 plt.show()
196
197 # ==== Improved Semi-discretization precision ====
198 # == Save results ==
199 # Check if there exists directory 'cache'.
200 if not os.path.exists('cache'):
201     os.makedirs('cache')
202
203 # Check if a file with stored results exists.
204 fname = os.path.join('cache', 'semi_precision.csv')
205 if os.path.exists(fname):
206     df = pd.read_csv(fname)
207 else:
208     # Generate random positive matrices.
209     rng = np.random.default_rng(seed=842463)
210     N = 3
211     B0 = rng.normal(scale=10, size=(N, N))
212     B1 = rng.normal(scale=10, size=(N, N))
213     # Set dimensions of the Tustin matrices.
214     approx = 2 * 2**np.arange(app_low, app_high)
215     errors = []
216     mus = []
217     final_times = []
218     for M in approx:
```

```
219         print(f'Approximation: {M}')
220         matrix = semi_discretization_matrix(B0, B1, M)
221         # Set function 1 as initial guess.
222         v0 = np.ones(M * N)
223         times = []
224         for _ in range(n_rep):
225             start = time.time()
226             mu = eigs(
227                 matrix,
228                 k = 1,
229                 v0 = v0,
230                 which = 'LM',
231                 return_eigenvectors = False
232             )
233             end = time.time()
234             times.append(end - start)
235         if min(times) != 0:
236             final_times.append(min(times))
237         else:
238             final_times.append(sum(times) / n_rep)
239         mu = np.log(mu) * (M - 1)
240         mu = np.array(mu)
241         error = [test_eigenvalue(B0, B1, e) for e in mu]
242         error = np.array(error)
243         errors.append(error)
244         mus.append(mu)
245     mus = np.stack(mus)
246     errors = np.stack(errors)
247     final_times = np.array(final_times)
248 
249     # Create df to store the results.
250     df = pd.DataFrame()
251     df['dim'] = approx * N
252     df['time (s)'] = final_times
253     # Add eigenvalues and errors to the df.
254     for i in range(mus.shape[1]):
255         df[f'eig_{i}'] = mus[:, i]
256     for i in range(mus.shape[1]):
257         df[f'error_{i}'] = errors[:, i]
258     # Save df to a csv file.
259     df.to_csv(fname, index=False, sep=',')
260 
261 
262 # == Plot results ==
263 # Set format for plots.
264 plt.rcParams.update(plt.rcParamsDefault)
265 plt.rcParams['text.usetex'] = True
266 plt.rcParams['font.size'] = 22
267 plt.rcParams['font.family'] = 'serif'
268 
269 # Set colors to identify the eigenvalues.
270 colors = ['maroon']
271 
272 fig, axs = plt.subplots()
```

```
axs.plot(
    np.log2(df['dim']), np.log10(df[f'error_{0}']), 'o-', label=f'eig {0}', color=colors[0])
axs.set_xlabel(r'$\log_2$(order of approximation)')
axs.set_ylabel(r'$\log_{10}$(error)')
axs.legend(loc='center left', bbox_to_anchor=(1, 0.5))

r = linregress(np.log2(df['dim']),
               np.log10(df['error_0']))
print(r)
print(f'Exponent: {r.slope / np.log10(2)}')

fig.set_size_inches(10, 8)
plt.show()
```

A.4 Comparisons

```
# ===== Imports =====
# Add path to module for stability
import os

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# ===== Compare times =====
# == Load data ==
# Load data from CSV files
data_dir = 'cache'
data_files = ['semi_precision.csv', 'tustin_stable_precision.csv', 'initial_semi_precision.csv']
df_s = pd.read_csv(os.path.join(data_dir, data_files[0]))
df_t = pd.read_csv(os.path.join(data_dir, data_files[1]))
df_s2 = pd.read_csv(os.path.join(data_dir, data_files[2]))

# Set format for plots.
plt.rcParams.update(plt.rcParamsDefault)
plt.rcParams['text.usetex'] = True
plt.rcParams['font.size'] = 22
plt.rcParams['font.family'] = 'serif'

fig, ax = plt.subplots()
ax.plot(
    np.log2(df_t['dim']), df_t['time (s)'], '-o', label='Tustin')
ax.plot(
    np.log2(df_s['dim']), df_s['time (s)'], '-o', label='Semi-discretization')
ax.plot(
    np.log2(df_s2['dim']), df_s2['time (s)'], '-o', label='Initial Semi-discretization')
ax.set_xlabel(r'$\log_2$(dimension)')
```

```
ax.set_ylabel(r'time (s)')
ax.legend(loc='upper left')

fig.set_size_inches(10, 10)
plt.show()
# ===== Compare precisions =====
# == Load data ==
# Load data from CSV files
data_dir = 'cache'
data_files = ['semi_precision.csv', 'tustin_stable_precision.
    csv', 'initial_semi_precision.csv']
df_s = pd.read_csv(os.path.join(data_dir, data_files[0]))
df_t = pd.read_csv(os.path.join(data_dir, data_files[1]))
df_s2 = pd.read_csv(os.path.join(data_dir, data_files[2]))

# Set format for plots.
plt.rcParams.update(plt.rcParamsDefault)
plt.rcParams['text.usetex'] = True
plt.rcParams['font.size'] = 22
plt.rcParams['font.family'] = 'serif'

fig, ax = plt.subplots()
ax.plot(
    np.log2(df_t['dim']), np.log10(df_t['error_0']), '-o',
    label='Tustin')
ax.plot(
    np.log2(df_s['dim']), np.log10(df_s['error_0']), '-o',
    label='Semi-discretization')
ax.plot(
    np.log2(df_s2['dim']),
    np.log10(df_s2['error_0']), '-o', label='Initial Semi-
    discretization')
ax.set_xlabel(r'$\log_2$(dimension)')
ax.set_ylabel(r'$\log_{10}$(error)')
ax.legend(loc='upper right')

fig.set_size_inches(10, 10)
plt.show()
```

Bibliography

[1] Insperger, T.; Stépán, G. Semi-discretization for time-delay systems. Stability and engineering applications. Applied Mathematical Sciences, 178. Springer, New York, 2011.

[2] Hayes, N. D. Roots of the transcendental equation associated with a certain difference-differential equation. J. London Math. Soc. 25 (1950), 226–232.

[3] Cushing, J. M. Time delays in single species growth models. J. Math. Biol. 4 (1977), no. 3.

[4] Verduyn Lunel, Sjoerd M. Spectral theory for delay equations. Systems, approximation, singular integral operators, and related topics (Bordeaux, 2000), 465–507, Oper. Theory Adv. Appl., 129, Birkhäuser, Basel, 2001.

[5] Pazy, A. Semigroups of linear operators and applications to partial differential equations. Applied Mathematical Sciences, 44. Springer-Verlag, New York, 1983.

[6] Osborn, John E. Spectral approximation for compact operators. Math. Comput. 29 (1975), 712–725.

[7] Ponce-Vanegas, Felipe; Irastorza, Markel. The Tustin method for approximating eigenvalues of delay systems. Still in progress.

[8] Duoandikoetxea, Javier. Fourier analysis. Translated and revised from the 1995 Spanish original by David Cruz-Uribe. Graduate Studies in Mathematics, 29. American Mathematical Society, Providence, RI, 2001.