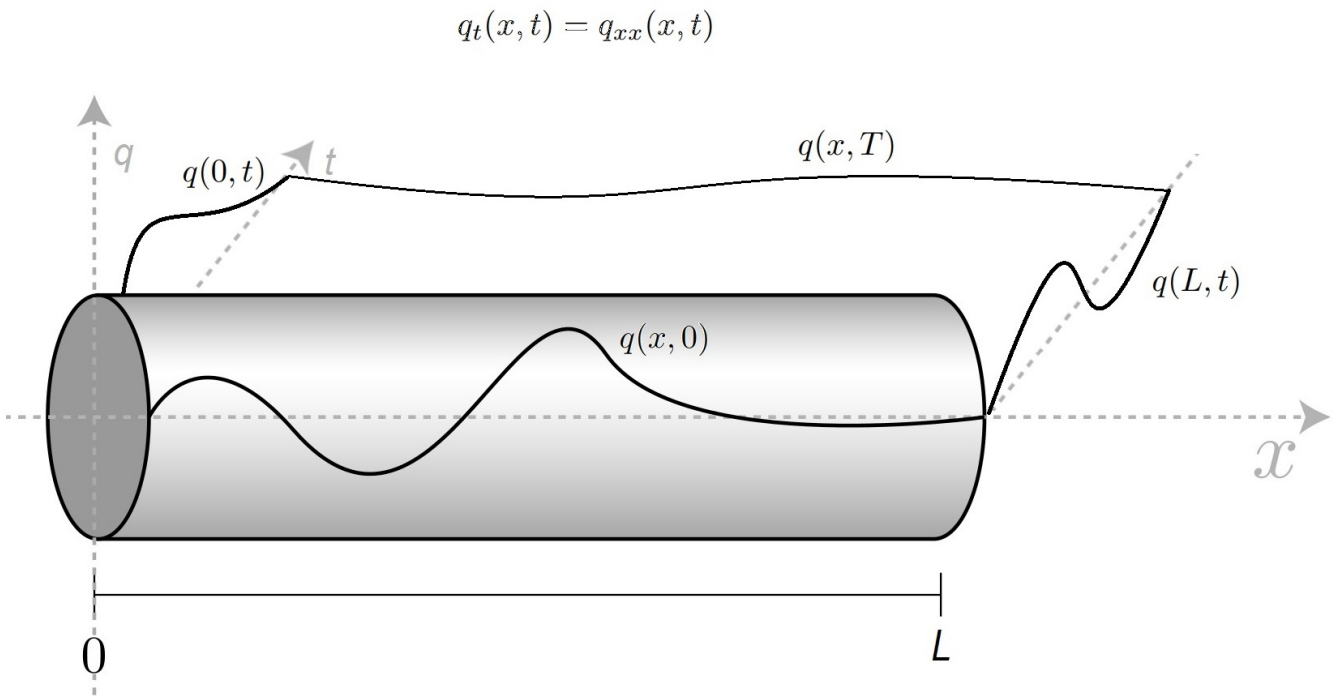
Two boundary value control algorithms

For the heat equation on the finite domain using the unified transform method

SC52135: S&C MSc Thesis

Thijs van Wijk

Delft University of Technology



TU Delft

Two boundary value control algorithms

For the heat equation on the finite domain using the unified transform method

by

Thijs van Wijk

Student Name	Student Number
Thijs van Wijk	4351509

Instructor: Dr.-Ing. S. Wahls
Project Duration: August, 2022 - July, 2023
Faculty: Faculty of Systems and Control, Delft University of Technology, Delft

Cover: (Modified version of) ”Idealized physical setting for heat conduction in a rod with homogeneous boundary conditions.” [2]
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Abstract

In this work we investigate two boundary-value control algorithms for the heat equation on the finite interval. The algorithms we discuss here are based on the unified transform method (UTM), a method invented by A. S. Fokas to solve boundary value problems of partial differential equations. We first inspect the boundary-value control algorithm presented in Kalimeris et al. [10]. This algorithm is originally constructed to find the Neumann boundary value on the right side for nullcontrol of the heat equation. In this work the algorithm has been expanded to allow arbitrary control objective, with either the Dirichlet or Neumann boundary values from both sides. Other improvements have also been made.

Furthermore, a second algorithm is constructed, which aims for a lower computational cost than the first algorithm. This second algorithm uses the same principles as the algorithm used in [10], but stems from another part of the derivation of the UTM. Both algorithms were tested with various control objectives, and show promising results.

Contents

1 Introduction

Partial differential equations (PDEs) form the basis of a large number of physical relations. It is therefore not uncommon that engineers (who are known for occasionally dealing with physics) encounter such a PDE. PDEs that only involve derivatives of one variable (commonly known as ordinary differential equations (ODEs)) are often solved easily enough, but when we move onto PDEs involving derivatives of two variables, things start to get more complex. When the PDE is simple enough, there will still be some methods to solve it. Think about separation of variables, or a Fourier/Laplace type transform. These common methods are often highly specialized for the PDE and its initial/boundary conditions, and they will break down with an increase in either the order of the PDE or the complexity of the boundary values. For most engineers, the solutions of these more complex PDEs will be to turn to any type of numerical solver.

Usually this works out fine, when the problem that needs solving just needs a solution from the given initial condition and boundary values. Sometimes, however, a problem requires a specific solution of a PDE, and we need to find the boundary values that give rise to that specific solution. To solve a control problem like this, we would normally find any math used to calculate the solution, and start working backwards, but this is impossible to do for any numerical solver. Reverse-engineering the Fourier/Laplace type transform methods could work, but this would only be possible for the handful of PDEs of low enough complexity.

This is where the the unified transform method (UTM) comes in. The UTM is a relatively new way to provide solutions to boundary value problems (BVPs) to a wide array of partial differential equations (PDEs). Invented by Athanassios S. Fokas, it uses the *local relation*[1] to describe the problem locally, then extrapolates this to the full domain to incorporate the boundary values. For the most basic case, linear evolution equations, the solution found by the UTM is a combination of two or more complex contour integrals. This integral representation, opposed to a solution based on infinite series often found by other methods, is generally advantageous for numerical computation [5] [4]. Besides the numerical advantages, the integral representation also clearly illustrates the relation between the known and unknown values.

Recently, Kalimeris et al. [10] used this relation between the known and unknown values in the UTM to construct an algorithm which approximates the *Neumann boundary values* needed for nullcontrol[2] of the *heat equation*. For this algorithm the boundary values (also called the input in analogy to control system engineering) is approximated into N basis functions. The solution given by the UTM is then evaluated at M points. A system of linear equations is then constructed which can be solved for the (approximated) boundary values. The results found in [10] are highly accurate and it it shown that any error in the results decreases exponentially with increased computational effort. But there are also some drawbacks in the algorithm as it is presented, primarily on the small range of functions it can be used for.

In this work, several additions to the Kalimeris et Al. algorithm will be proposed. These additions aim to contain the control input and expand the range of initial and final conditions that the algorithm

[1] Italicized terms will be defined later.

[2] controlling the system to zero

can work with. The algorithm is also generalized to work with boundary conditions on both sides, and the option for control with *Dirichlet boundary values* is added. Besides mere additions, a second algorithm will also be proposed. This second algorithm takes inspiration from [10] and is also based on the methods of the UTM, but is based in the complex plane instead of the real line. This algorithm is developed with computational efficiency in mind, with the hope that it can be as accurate as the Kalimeris et Al. algorithm.

We will start in Chapter 2 with the derivation of the solution of the heat equation using the UTM. In Chapter 3 we will discuss the Kalimeris et al. algorithm and propose our additions. The second algorithm will be proposed in Chapter 4. In Chapter 5 tests will be performed to compare the effectiveness of the algorithms. Finally we will conclude our findings in Chapter 6. The code used for this work can be found in the Appendix.

Finally we note that in this work we will not address controllability. Instead we will approach any control problem with the practical assumption that the set of reachable states is at least dense in L^2[3]. This means that if the desired state should be unreachable, there should be a reachable state that is close enough to this desired state.

[3]The set of square-integrable functions.

2

The unified transform method for the heat equation on the finite domain

In this chapter the basic principles of the UTM will be demonstrated for the heat equation on the finite domain,

$$q_t(x,t) = q_{xx}(x,t), \qquad (x,t) \in \Omega, \tag{2.1a}$$

$$\Omega = \{0 \leq x \leq L,\ 0 \leq t \leq T\}. \tag{2.1b}$$

Here we also defined Ω, the domain of q in x and t. The domain is illustrated in figure 2.1, along with $g_0(t)$ and $g_1(t)$, $h_0(t)$ and $h_1(t)$, $q_0(x)$, and $q_T(x)$, which are shorthand terms for the boundary values and the initial and final conditions that we use for convenience:

$$\begin{aligned} q_0(x) &:= q(x,0), \quad q_T(x) := q(x,T) && x \in (0,L), \\ g_0(t) &:= q(0,t), \quad g_1(t) := q_x(0,t), && t \in (0,T), \\ h_0(t) &:= q(L,t), \quad h_1(t) := q_x(L,t), && t \in (0,T). \end{aligned}$$

It is assumed all of these terms are sufficiently smooth.

Normally not all of the boundary values are given. If $g_0(t)$ and $h_0(t)$ are given, one speaks of the *Dirichlet problem*. These boundary values are also known as the Dirichlet boundary values. When $g_1(t)$ and $h_1(t)$, the Neumann boundary values, are given, one speaks of the *Neumann problem*.

We start with the derivation of the general solution is Section 2.1, which we will use to derive the Neumann solution in Section 2.2, and the Dirichlet solution in Section 2.3.

We note again that the UTM can be applied to a large class of linear evolution PDEs, but the full derivation of this would be overly complex in a work where only the heat equation is used. The interested reader can find the derivation for the arbitrary linear evolution PDE in either [3], [16], or Chapters 1 and 2 of [6], ordered from more accessible to more in depth.

2.1. Derivation of the general solution

Our derivation will start with the so called *local relation* [6]:

$$(e^{-ikx+w(k)t}q)_t - (e^{-ikx+w(k)t}(q_x + ikq))_x = 0, \qquad k \in \mathbb{C}, \tag{2.2}$$

where k is some complex variable. The local relation describes how $q(x,t)$ acts locally, describing an equilibrium between some ratio of the partial derivatives in the x and t directions. It can be derived from the fact that (2.1) admits the one parameter family of solutions e^{ikx-k^2t} and from the adjoint of (2.1), in Chapter 9 of [6].

The next step is to integrate (2.2) over the entire domain of both x and t. The equation is now in such a form that allows us to use *Green's Theorem* on it. This theorem relates double integrals of expression in the form of (2.2) to a contour integral around the domain of integration:

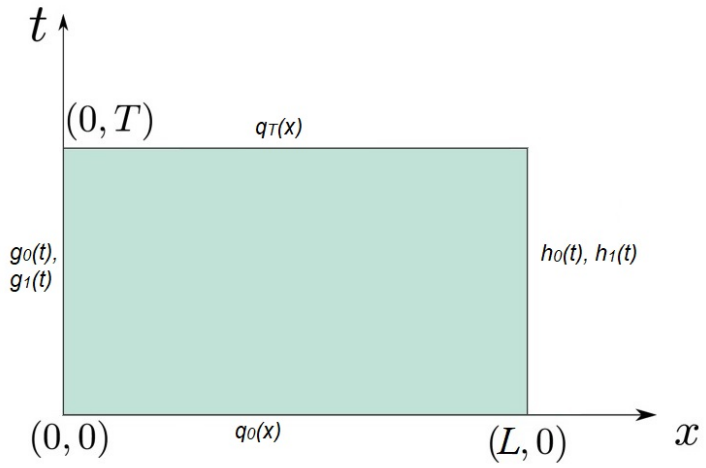


Figure 2.1: The domain of $q(x,t)$

Green's Theorem 1 ([14])**.** Let $P:(x,y)\to\mathbb{R}$ and $Q:(x,y)\to\mathbb{R}$ have continuous partial derivatives on an open region $D\subseteq\mathbb{R}^2$, and C is a positively oriented, piecewise-smooth, simple closed curve which is the boundary of D, then:

$$\int_C P(x,y)dx+Q(x,y)dy=\iint_D(\frac{\partial Q(x,y)}{\partial x}-\frac{\partial P(x,y)}{\partial y})dxdy$$

This will be used to transform the equation from a local one, to one relating the boundaries of the domain with each other (under certain transforms):

$$\begin{aligned}
&\iint_\Omega\left[(e^{-ikx+k^2t}q(x,t))_t-(e^{-ikx+k^2t}(q_x(x,t)+ikq(x,t)))_x\right]dtdx && \text{Green's Theorem}\\
=&\oint_{\partial\Omega}\left[e^{-ikx+k^2t}q(x,t)dx+(e^{-ikx+k^2t}(q_x(x,t)+ikq(x,t)))dt\right] && \text{separate boundary}\\
=&\int_0^L e^{-ikx}q(x,0)dx+\int_0^T e^{-ikL+k^2t}(q_x(L,t)+ikq(L,t))dt\\
&-\int_0^L e^{-ikx+k^2T}q(x,T)dx-\int_0^T e^{k^2t}(q_x(0,t)+ikq(0,t))dt && \text{employ transform notation}\\
=&\hat{q}_0(k)+e^{-ikL}[\tilde{h}_1(k^2)+ik\tilde{h}_0(k^2)]-e^{k^2T}\hat{q}_T(k)-[\tilde{g}_1(k^2)+ik\tilde{g}_0(k^2)].
\end{aligned}$$

When writing the contour integral as separate integrals in the 'separate boundary' step, we use the fact that either dx or dt is zero on those parts of the boundaries, so they can be removed from the integral. In the 'employ transform notation' step we have used both the *Fourier transform* and the *T-transform*:

Definition. The *Fourier transform* of a function $f(x):[0,L]\to\mathbb{R}$ is defined as

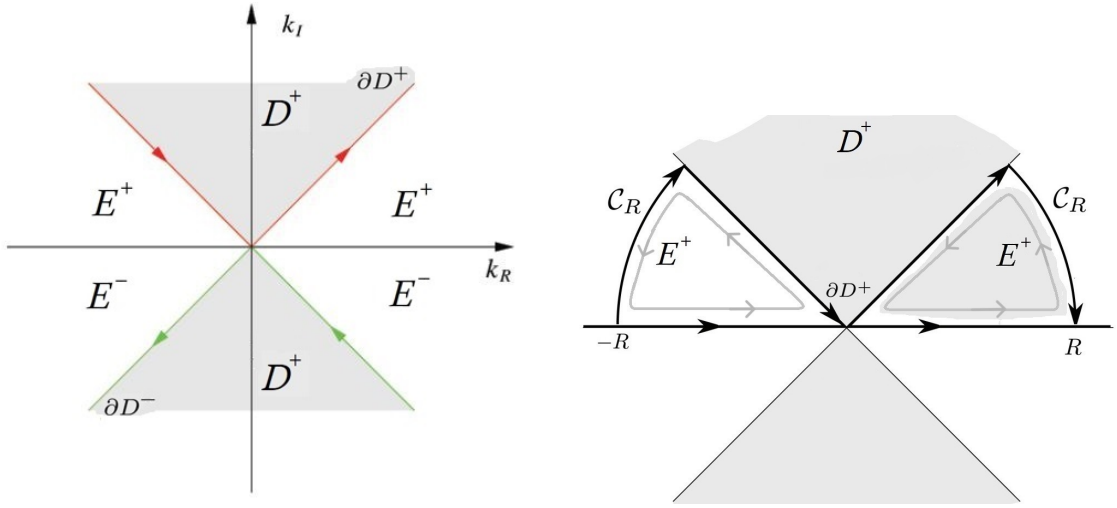
$$\hat{f}(k):=\int_0^L e^{-ikx}f(x)dx,\qquad k\in\mathbb{C},$$

where $\hat{f}:\mathbb{C}\to\mathbb{C}$ is the Fourier transform of $f(x)$.

Definition. the *T-transform* or time transform of a function $f(t):[0,T]\to\mathbb{R}$ is defined as

$$\tilde{f}(k^2):=\int_0^T e^{k^2t}f(t)dt,\qquad k\in\mathbb{C},$$

with $\tilde{f}(k^2):\mathbb{C}\to\mathbb{C}$.



(a) Domains D and E and the contours $\partial D^{\pm}$ **(b)** Visual representation of the contour deformation to ∂D^+

Figure 2.2: Complex domain integration contours

This pair of transforms are the characteristic transforms of the heat equation. The hat $\hat{}$ symbol denotes the Fourier transform of a function and the tilde $\widetilde{}$ symbol denotes *T-transform* of a function.

We slightly rewrite the last obtained result to get the *global relation* (GR), an equation which holds all (transformed) boundaries of our domain,

$$\hat{q}_0(k) - \widetilde{g}(k) + e^{-ikL}\widetilde{h}(k) = e^{k^2T}\hat{q}_T(k), \qquad k \in \mathbb{C}, \tag{2.3}$$

where

$$\begin{aligned} \widetilde{g}(k) &:= ik\widetilde{g}_0(k^2) + \widetilde{g}_1(k^2), \\ \widetilde{h}(k) &:= ik\widetilde{h}_0(k^2) + \widetilde{h}_1(k^2). \end{aligned} \tag{2.4}$$

From the GR, (2.3), we divide by e^{k^2T} and apply the inverse Fourier transform to find a solution

$$\begin{aligned} q(x,T) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\hat{q}_0(k)dk - \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\widetilde{g}(k)dk \\ + \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-ik(L-x)-k^2T}\widetilde{h}(k)dk. \end{aligned}$$

While this is a solution for $q(x,T)$ there are still a few more steps left in the UTM. That's because the found solution still contains all 4 boundary value terms, while normally only 2 are given. This is why the last step of the UTM is to cancel out two of these unknown boundary values (as performed in section 2.2 for the Neumann case). To prepare for this, we will deform the contours of integration of the latter two integrals (the ones involving $\tilde{g}(k)$ and $\tilde{h}(k)$.

We define the domains D^+, D^-, E^+, and E^-, which partition the complex k plane:

$$D^+ = \{k \in \mathbb{C}, \quad \text{Re}\, k^2 < 0, \quad \text{Im}\, k > 0\}, \tag{2.5a}$$
$$D^- = \{k \in \mathbb{C}, \quad \text{Re}\, k^2 < 0, \quad \text{Im}\, k < 0\}, \tag{2.5b}$$
$$E^+ = \{k \in \mathbb{C}, \quad \text{Re}\, k^2 > 0, \quad \text{Im}\, k > 0\}, \tag{2.5c}$$
$$E^- = \{k \in \mathbb{C}, \quad \text{Re}\, k^2 > 0, \quad \text{Im}\, k < 0\}. \tag{2.5d}$$

We also define ∂D^+ and ∂D^- as the boundaries of D^+ and D^-, respectively, in a counterclockwise direction. These can all be seen if figure 2.2a.

We also introduce Cauchy's integral theorem and Jordan's lemma, which are the final parts needed for the contour deformation:

Cauchy's integral Theorem 2 ([1])**.** If a function $f(z) : \mathbb{C} \to \mathbb{C}$ is analytic at all points interior to and on a simple closed contour C, then

$$\int_C f(z)dz = 0.$$

Definition. A function $f(z) : \mathbb{C} \to \mathbb{C}$ is *analytic* in an open set S if it has a derivative everywhere in that set. It is analytic in a point if it is analytic in some neighborhood around that point. [1]

Jordan's Lemma 3 ([1])**.** Suppose that

1. a function $f(z) : \mathbb{C} \to \mathbb{C}$ is analytic at all points in the upper half plane Im $z \geq 0$ that are exterior to a circle $|z| = R_0$;
2. C_R denotes a bigger semicircle $z = Re^{i\theta}$ where $0 \leq \theta \leq \pi$ and $R > R_0$
3. for all points z on C_R, there is a positive constant M_R such that

$$|f(z)| \leq M_R \text{ and } \lim_{R\to\infty} M_R = 0.$$

Then, for every positive constant a:

$$\lim_{R\to\infty} \int_{C_R} f(z)e^{iaz}dz = 0.$$

With all necessary thing defined, we now start with the contour deformation. We will first look at the integral $\int_{-\infty}^{\infty} e^{ikx-k^2T}\widetilde{g}(k)dk$. Specifically, we will look at a part of this integral first, namely a part distance R away from the origin, $\int_{-R}^{R} e^{ikx-k^2T}\widetilde{g}(k)dk$. This integral is equal to a circular arc C_R from $-R$ to D^+, a part of ∂D^+ (from $|R|$ to $|R|$), a mirrored circular arc back to the real line, and two counterclockwise contours around E^+ up to R, all with the same integrand. This is visualized in figure 2.2b, where we can see that all contours besides the one on the real line do indeed cancel out. As we let $R \to \infty$, we find:

$$\int_{-\infty}^{\infty} e^{ikx-w(k)t}\tilde{g}(k)dk = \left(\sum \oint_{E^+} + \sum \lim_{R\to\infty} \int_{C_R} + \int_{\partial D^+}\right) e^{ikx-w(k)t}\tilde{g}(k)dk. \tag{2.6}$$

The integrand $e^{ikx-k^2T}\widetilde{g}(k)$ in domain is analytic in E^+, meaning we can use Cauchy's integral theorem to show that the contour integrals around E^+ in (2.6) are zero. The integrand is also bounded on E^+, as $\widetilde{g}(k)$ is also bounded due to the finite domain of the transform, and $T > 0$ and $x \geq 0$, so the exponential is bounded for Re$k^2 \geq 0$ and Im$k \geq 0$, or on E^+. Furthermore, the integrand decays to zero as $|k| \to \infty$, which can be shown via integration by parts, [6]. This means we can use Jordan's lemma to show that the circular contours C_R are also zero. We're left with:

$$\int_{-\infty}^{\infty} e^{ikx-w(k)t}\tilde{g}(k)dk = \int_{\partial D^+} e^{ikx-w(k)t}\tilde{g}(k)dk. \tag{2.7}$$

The same can be done with the third integral in (2.1), but now on E^- (as $L - x \geq 0$, so this is bounded for Im$k \leq 0$). In this calculation an extra minus sign arises from the way we define ∂D^-. With both contour deformation found, we get

$$\begin{aligned} q(x,T) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\hat{q}_0(k)dk - \frac{1}{2\pi}\int_{\partial D^+} e^{ikx-k^2T}\widetilde{g}(k)dk \\ -\frac{1}{2\pi}\int_{\partial D^-} e^{-ik(L-x)-k^2T}\widetilde{h}(k)dk. \end{aligned} \tag{2.8}$$

This the general solution to the heat equation. Since it contains all the boundary values in $\tilde{g}(k)$ and $\tilde{h}(k)$, it can't be used for any specific problem yet. In the following sections the Neumann, and the Dirichlet problem will be derived.

2.2. Derivation of the Neumann problem of the heat equation

In this section we derive the solution of the Neumann problem of the heat equation on the finite interval. The derivation will start from both the general solution (2.8) and the global relation (2.3) of the heat equation on the finite interval. From there we follow the last steps of the UTM to come to the final solution.

The first step is isolating and removing the unknown boundary values from (2.8). For the Neumann problem, these are $\tilde{g}_0(k^2)$ and $\tilde{h}_0(k^2)$ (which are contained in $\tilde{g}$(k) and $\tilde{h}$(k) (2.4)). To do this we create system of equations by using the transformation $k \to -k$ on the global relation. The variables we want to isolate are functions of k^2 and thus invariant to this transformation:

$$\hat{q}_0(-k) + ik\widetilde{g}_0(k^2) - \widetilde{g}_1(k^2) - ike^{ikL}\widetilde{h}_0(k^2) + e^{ikL}\widetilde{h}_1(k^2) = e^{k^2T}\hat{q}_T(-k), \qquad k \in \mathbb{C}, \tag{2.9}$$

The GR, (2.3), and the GR with the transformation $k \to -k$ (2.9) now form a system of two equations from which we separate $\tilde{g}_0(k^2)$ and $\tilde{h}_0(k^2)$ from the rest of the terms:

$$\begin{bmatrix} -1 & e^{-ikL} \\ 1 & -e^{ikL} \end{bmatrix} \begin{bmatrix} ik\tilde{g}_0(k^2) \\ ik\tilde{h}_0(k^2) \end{bmatrix} = \begin{bmatrix} -\hat{q}_0(k) + \tilde{g}_1(k^2) - e^{-ikL}\tilde{h}_1(k^2) + e^{k^2T}\hat{q}_T(k) \\ -\hat{q}_0(-k) + \tilde{g}_1(k^2) - e^{ikL}\tilde{h}_1(k^2) + e^{k^2T}\hat{q}_T(-k) \end{bmatrix} =: \begin{bmatrix} N_0(k) \\ N_0(-k) \end{bmatrix},$$

where we introduce the encompassing function $N_0(k)$ for the ease of notation. From here we take the inverse of the leftmost matrix and find

$$\begin{aligned} \begin{bmatrix} ik\tilde{g}_0(k^2) \\ ik\tilde{h}_0(k^2) \end{bmatrix} &= \frac{1}{e^{ikL} - e^{-ikL}} \begin{bmatrix} -e^{ikL} & -e^{-ikL} \\ -1 & -1 \end{bmatrix} \begin{bmatrix} N_0(k) \\ N_0(-k) \end{bmatrix} \\ &= \frac{1}{e^{ikL} - e^{-ikL}} \begin{bmatrix} -e^{ikL}N_0(k) - e^{-ikL}N_0(-k) \\ -N_0(k) - N_0(-k) \end{bmatrix}. \end{aligned}$$

We note that when $k = n2\pi$, for integer n, the determinant of the system is zero. This will be dealt with accordingly when it comes up. We plug these values into $\widetilde{g}(k)$ and $\widetilde{h}(k)$, we find

$$\begin{aligned} \widetilde{g}(k) &= ik\widetilde{g}_0(k^2) + \widetilde{g}_1(k^2) \\ &= \widetilde{g}_1(k^2) + \frac{-e^{ikL}N_0(k) - e^{-ikL}N_0(-k)}{e^{ikL} - e^{-ikL}} \\ &= \frac{\cancel{e^{ikL}\widetilde{g}_1(k^2)} - e^{-ikL}\widetilde{g}_1(k^2)}{e^{ikL} - e^{-ikL}} - \frac{-e^{ikL}\hat{q}_0(k) + \cancel{e^{ikL}\widetilde{g}_1(k^2)} - \tilde{h}_1(k^2) + e^{ikL+k^2T}\hat{q}_T(k)}{e^{ikL} - e^{-ikL}} \\ &\qquad - \frac{-e^{-ikL}\hat{q}_0(-k) + e^{-ikL}\tilde{g}_1(k^2) - \tilde{h}_1(k^2) + e^{-ikL+k^2T}\hat{q}_T(-k)}{e^{ikL} - e^{-ikL}} \\ &= \frac{-e^{-ikL}2\widetilde{g}_1(k^2) + 2\widetilde{h}_1(k^2) + e^{ikL}\hat{q}_0(k) - e^{ikL+k^2T}\hat{q}_T(k) + e^{-ikL}\hat{q}_0(-k) - e^{-ikL+k^2T}\hat{q}_T(-k)}{e^{ikL} - e^{-ikL}} \end{aligned} \tag{2.10}$$

and

$$\begin{aligned} \widetilde{h}(k) &= ik\widetilde{h}_0(k^2) + \widetilde{h}_1(k^2) \\ &= \widetilde{h}_1(k^2) + \frac{-N_0(k) - N_0(-k)}{e^{ikL} - e^{-ikL}} \\ &= \frac{e^{ikL}\widetilde{h}_1(k^2) \cancel{-e^{-ikL}\widetilde{h}_1(k^2)}}{e^{ikL} - e^{-ikL}} - \frac{-\hat{q}_0(k) + \tilde{g}_1(k^2) \cancel{-e^{-ikL}\tilde{h}_1(k^2)} + e^{k^2T}\hat{q}_T(k)}{e^{ikL} - e^{-ikL}} \\ &\qquad - \frac{-\hat{q}_0(-k) + \tilde{g}_1(k^2) - e^{ikL}\tilde{h}_1(k^2) + e^{k^2T}\hat{q}_T(-k)}{e^{ikL} - e^{-ikL}} \\ &= \frac{-2\widetilde{g}_1(k^2) + 2e^{ikL}\widetilde{h}_1(k^2) + \hat{q}_0(k) - e^{k^2T}\hat{q}_T(k) + \hat{q}_0(-k) - e^{k^2T}\hat{q}_T(-k)}{e^{ikL} - e^{-ikL}}. \end{aligned} \tag{2.11}$$

Now that we have found expressions for $\widetilde{g}(k)$ and $\widetilde{h}(k)$ without the unknown boundary values we can substitute them into the general solution (2.8). This, however, brings a the new problem that our

solution $q(x,T)$ now depends on its own Fourier transform:

$$q(x,T) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\hat{q}_0(k)dk \quad \cdots \tag{2.12}$$

$$-\frac{1}{2\pi}\int_{\partial D^+}\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[-e^{-ikL}2\widetilde{g}_1(k^2)+2\widetilde{h}_1(k^2)+e^{ikL}\hat{q}_0(k)+e^{-ikL}\hat{q}_0(-k)-e^{k^2T}(e^{ikL}\hat{q}_T(k)+e^{-ikL}\hat{q}_T(-k))\right]dk$$

$$-\frac{1}{2\pi}\int_{\partial D^-}\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[-e^{-ikL}2\widetilde{g}_1(k^2)+2\widetilde{h}_1(k^2)+e^{-ikL}\hat{q}_0(k)+e^{-ikL}\hat{q}_0(-k)-e^{-ikL+k^2T}(\hat{q}_T(k)+\hat{q}_T(-k))\right]dk.$$

In the following part, we will specifically look at the parts of the integrals that contain the terms $\hat{q}_T(k)$ and $\hat{q}_T(-k)$:

$$\frac{1}{2\pi}\int_{\partial D^+}\frac{e^{ik(x+L)}\hat{q}_T(k)+e^{ik(x-L)}\hat{q}_T(-k)}{e^{ikL}-e^{-ikL}}dk+\frac{1}{2\pi}\int_{\partial D^-}\frac{e^{ik(x-L)}(\hat{q}_T(k)+\hat{q}_T(-k))}{e^{ikL}-e^{-ikL}}dk \tag{2.13}$$

The expression above contains poles at $k=0$. Were it not for these poles, we could argue that the total value of these integrals is zero (the full argument of this will follow later). To do this we first need to find the contribution of these poles, for which we ave to go over some complex analysis theory first.

Cauchy's integral theorem states that any contour integral of a function around a region in where that function is fully analytic, is zero, where the function also has to be analytic on the contour. A theorem related to this is Cauchy's residue theorem [8], which states that a contour integral of a function around a region where the function is analytic except on a number of poles[1] is equal to $2i\pi$ times the sum of *residues* of those poles (where the function still has to be analytic on the contour). The residue of a pole of a function is therefore defined by the constant value[2] obtained by performing a contour integration around it, divided by $2\pi i$. But despite being defined as such, there are other ways to calculate a residue, for example, for a function $f(z)=\frac{h(z)}{k(z)}:\mathbb{C}\to\mathbb{C}$ with a simple pole at z_0, we can find it residue as [8]

$$\text{res}\{f(z);z=z_0\}=\frac{h(z)}{dk(z)/dz}. \tag{2.14}$$

However, our problem is that we're integrating over, and not around, a pole. This is where the indentation lemma comes in [13]. As integrating over a pole is mathematically ambiguous, it instead deforms the contour of integration to a tiny circular arc around the pole, and then calculates the contribution of that arc. Functionally it works the same as Cauchy's residue theorem, but instead of multiplying the residue with $2i\pi$ we multiply with i times the arc that the contour makes over the pole:

Indentation Lemma 4 ([13])**.** Let $f(z):\mathbb{C}\to\mathbb{C}$ have a simple pole at z_0 with a residue res$\{f(z);z=z_0\}$. Then

$$\lim_{\epsilon\to 0}\int_{c_e} f(z)dz = i(\beta-\alpha)\text{res}\{f(z);z=z_0\},$$

where c_e denotes the circular arc $\theta\to z_0+\epsilon e^{i\theta}$, $\alpha\le\theta\le\beta$ around the pole.

Thus, to calculate the contribution of the integrals in (2.13), we deform the contours involving ∂D^+ and ∂D^- to exclude a tiny circular arc from the origin. We will call these contours ∂D_c^+ and ∂D_c^-, respectively. They can be seen in figure 2.3. We define $\partial D_c^\pm$ as $\partial D_c^+ := \lim_{\epsilon\to 0}\partial D^+\backslash\{\epsilon e^{i\theta}\}$ for $\frac{\pi}{4}\le\theta\le\frac{3\pi}{4}$ and $D_c^- := \lim_{\epsilon\to 0}\partial D^-\backslash\{\epsilon e^{i\theta}\}$ for $\frac{5\pi}{4}\le\theta\le\frac{7\pi}{4}$.

We start the residue calculation with the ∂D^- integral from (2.13). We use the fact that $e^{ikx-k^2T}\tilde{h}(k)$ is analytic in the full k plane, this means the pole at $k=0$ was introduced by removing the unknown boundary values in (2.11). This also means its 'residue' at $k=0$ is Res$\{e^{-ik(L-x)-k^2T}\tilde{h}(k),k=0\}=0$[3].

[1] The theorem also extends to other types of (isolated) singularities, not just poles. As these are not needed for this explanation they will be ignored for simplicity.

[2] Or a value independent of the integration variable in the case of multiple variables

[3] Technically, this is not a residue as there is no pole. What we mean by residue here is the value we would find if we apply a contour integral of this function around this point (divided by $2i\pi$).

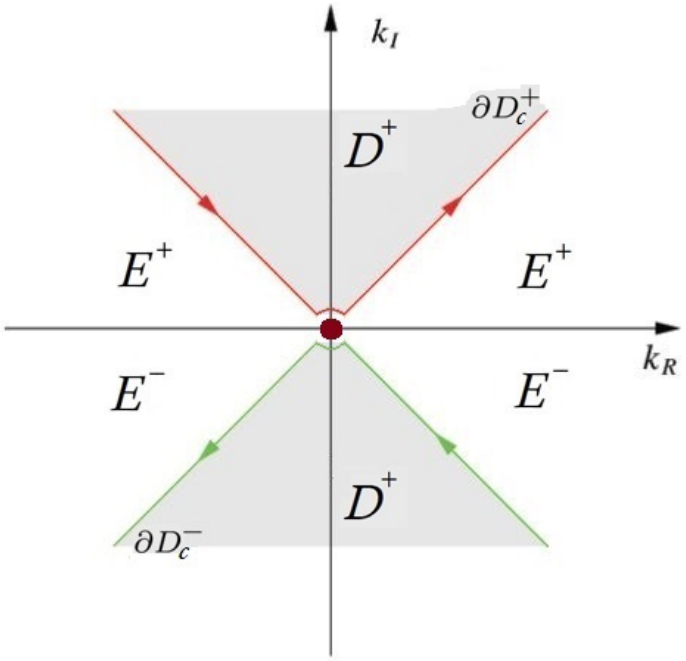


Figure 2.3: Contours $\partial D_c^{\pm}$. Similar to contours $\partial D^{\pm}$ in figure 2.2a, but with an indentation around $k = 0$.

Now we plug in the expression for $\tilde{h}(k)$ found in (2.11) to find

$$\text{Res}\{\frac{e^{-ik(L-x)}[\hat{q}_T(k)+\hat{q}_T(-k)]}{e^{ikL}-e^{-ikL}}, k=0\} = \text{Res}\{\frac{e^{-ik(L-x)-k^2T}[-2\widetilde{g}_1(k^2)+2e^{ikL}\widetilde{h}_1(k^2)+\hat{q}_0(k)+\hat{q}_0(-k)]}{e^{ikL}-e^{-ikL}}, k=0\}$$
$$= \frac{\tilde{g}_1(0)-\tilde{h}_1(0)-\hat{q}_0(0)}{iL}, \tag{2.15}$$

where we used the method of calculating residues for simple poles in (2.14). We do the same with the ∂D^+ we find that is has a similar residue:

$$\text{Res}\{\frac{e^{ik(x+L)}\hat{q}_T(k)+e^{ik(x-L)}\hat{q}_T(-k)}{e^{ikL}-e^{-ikL}}, k=0\}$$
$$=\text{Res}\{\frac{e^{ikx-k^2T}[-2e^{-ikL}\widetilde{g}_1(k^2)+2\widetilde{h}_1(k^2)+e^{ikL}\hat{q}_0(k)+e^{-ikL}\hat{q}_0(-k)]}{e^{ikL}-e^{-ikL}}, k=0\} = \frac{-\tilde{g}_1(0)+\tilde{h}_1(0)+\hat{q}_0(0)}{iL}.$$

Then, the indentation lemma tells us that the contribution of the pole to the integral over ∂D_- in (2.13) is the found residue times i times the angle of the circular arc over the pole, which is $\frac{\pi}{2}$. The total contribution of both integrals is

$$\frac{1}{2\pi}2i\frac{\pi}{2}\frac{-\tilde{g}_1(0)+\tilde{h}_1(0)+\hat{q}_0(0)}{iL} = \frac{\tilde{g}_1(0)-\tilde{h}_1(0)-\hat{q}_0(0)}{2L},$$

where the factor $\frac{1}{2\pi}$ is the constant before the integrals (2.13), and the minus sign arises from the clockwise direction of the contours. We can now write (2.13) as

$$\frac{\tilde{g}_1(0)-\tilde{h}_1(0)-\hat{q}_0(0)}{2L}+\frac{1}{2\pi}\int_{\partial D_c^+}\frac{e^{ik(x+L)}\hat{q}_T(k)+e^{ik(x-L)}\hat{q}_T(-k)}{e^{ikL}-e^{-ikL}}dk+\frac{1}{2\pi}\int_{\partial D_c^-}\frac{e^{ik(x-L)}(\hat{q}_T(k)+\hat{q}_T(-k))}{e^{ikL}-e^{-ikL}}dk,$$

where we see that the integrands are bounded and analytic on $D_c^{\pm}$ (the regions $D^{\pm}$ excluding the regions around the pole), and decay to zero as $k \to \infty$, thus we can use Cauchy's integral theorem

with Jordan's lemma to see that any other contribution of these terms is zero. We find:

$$\begin{aligned} q(x,T) &= \frac{\tilde{g}_1(0) - \tilde{h}_1(0) - \hat{q}_0(0)}{2L} + \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\hat{q}_0(k)dk \\ -\frac{1}{2\pi}\int_{\partial D^+} &\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[-e^{-ikL}2\widetilde{g}_1(k^2) + 2\widetilde{h}_1(k^2) + e^{ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk \\ -\frac{1}{2\pi}\int_{\partial D^-} &\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[-e^{-ikL}2\widetilde{g}_1(k^2) + 2\widetilde{h}_1(k^2) + e^{-ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk. \end{aligned} \tag{2.16}$$

This equation only has known quantities on its right hand side, and is therefore a valid solution to our problem.

Alternatively, instead of only deforming the parts of the integral with $\hat{q}_t(\pm k)$, we can deform the full integrals of (2.8) before cancelling out the unknown boundary values. Since we know $\widetilde{g}$ and $\widetilde{h}$ are analytic, there will be no constant contribution from any poles. Afterwards we use $\widetilde{g}$ and $\widetilde{h}$ from (2.10) and (2.11), respectively, and then set the parts of the integrals containing $\hat{q}_T(\pm k)$ to zero.

This alternative formulation is

$$\begin{aligned} q(x,T) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\hat{q}_0(k)dk \\ -\frac{1}{2\pi}\int_{\partial D_c^+} &\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[-e^{-ikL}2\widetilde{g}_1(k^2) + 2\widetilde{h}_1(k^2) + e^{ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk \\ -\frac{1}{2\pi}\int_{\partial D_c^-} &\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[-e^{-ikL}2\widetilde{g}_1(k^2) + 2\widetilde{h}_1(k^2) + e^{-ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk. \end{aligned} \tag{2.17}$$

2.3. Derivation of the Dirichlet solution

We start the derivation of the Dirichlet solution from the system of equations formed by both version of the GR, the normal one (2.3) and the one with the $k \to -k$ transformation (2.9). Once again we isolate the unknown boundary values to eliminate them later. For the Dirichlet problem, these are $\tilde{g}_1(k^2)$ and $\tilde{h}_1(k^2)$:

$$\begin{bmatrix} -1 & e^{-ikL} \\ -1 & e^{-ikL} \end{bmatrix}\begin{bmatrix} \tilde{g}_1(k^2) \\ \tilde{h}_1(k^2) \end{bmatrix} = \begin{bmatrix} -\hat{q}_0(k) + ik\tilde{g}_0(k^2) - ike^{-ikL}\tilde{h}_0(k^2) + e^{k^2T}\hat{q}_T(k) \\ -\hat{q}_0(-k) - ik\tilde{g}_0(k^2) + ike^{ikL}\tilde{h}_0(k^2) + e^{k^2T}\hat{q}_T(-k) \end{bmatrix} =: \begin{bmatrix} N_1(k) \\ N_1(-k) \end{bmatrix},$$

where we define the encompassing function $N_1(k)$ for ease of notation. To solve for $\tilde{g}_1(k^2)$ and $\tilde{h}_1(k^2)$ we take the inverse of the leftmost matrix:

$$\begin{aligned} \begin{bmatrix} \tilde{g}_1(k^2) \\ \tilde{h}_1(k^2) \end{bmatrix} &= \frac{1}{e^{-ikL}-e^{ikL}}\begin{bmatrix} e^{ikL} & -e^{-ikL} \\ 1 & -1 \end{bmatrix}\begin{bmatrix} N_1(k) \\ N_1(-k) \end{bmatrix} \\ &= \frac{1}{e^{-ikL}-e^{ikL}}\begin{bmatrix} e^{ikL}N_1(k) - e^{-ikL}N_1(-k) \\ N_1(k) - N_1(-k) \end{bmatrix}. \end{aligned}$$

These values are then plugged into $\widetilde{g}(k)$ and $\widetilde{h}(k)$ in (2.4)

$$\begin{aligned} \widetilde{g}(k) &= ik\widetilde{g}_0(k^2) + \widetilde{g}_1(k^2) \\ &= ik\widetilde{g}_0(k^2) + \frac{e^{ikL}N_1(k) - e^{-ikL}N_1(-k)}{e^{-ikL}-e^{ikL}} \\ &= \frac{ike^{-ikL}\widetilde{g}_0(k^2) - \cancel{ike^{ikL}\widetilde{g}_0(k^2)}}{e^{-ikL}-e^{ikL}} + \frac{-e^{ikL}\hat{q}_0(k) + \cancel{ike^{ikL}\widetilde{g}_0(k^2)} - ik\tilde{h}_0(k^2) + e^{ikL+k^2T}\hat{q}_T(k)}{e^{-ikL}-e^{ikL}} \\ &\qquad - \frac{-e^{-ikL}\hat{q}_0(-k) - ike^{-ikL}\tilde{g}_0(k^2) + ik\tilde{h}_0(k^2) + e^{-ikL+k^2T}\hat{q}_T(-k)}{e^{-ikL}-e^{ikL}} \\ \widetilde{g}(k) &= \frac{2ike^{-ikL}\widetilde{g}_0(k^2) - 2ik\widetilde{h}_0(k^2) - e^{ikL}\hat{q}_0(k) + e^{ikL+k^2T}\hat{q}_T(k) + e^{-ikL}\hat{q}_0(-k) - e^{-ikL+k^2T}\hat{q}_T(-k)}{e^{-ikL}-e^{ikL}} \end{aligned} \tag{2.18}$$

,

and

$$\begin{aligned}
\widetilde{h}(k) &= ik\widetilde{h}_0(k^2) + \widetilde{h}_1(k^2) \\
&= ik\widetilde{h}_0(k^2) + \frac{N_1(k) - N_1(-k)}{e^{-ikL} - e^{ikL}} \\
&= \frac{\cancel{ike^{-ikL}\widetilde{h}_0(k^2)} - ike^{ikL}\widetilde{h}_0(k^2)}{e^{-ikL} - e^{ikL}} + \frac{-\hat{q}_0(k) + ik\tilde{g}_0(k^2) - \cancel{ike^{-ikL}\tilde{h}_0(k^2)} + e^{k^2T}\hat{q}_T(k)}{e^{-ikL} - e^{ikL}} \\
&\qquad - \frac{-\hat{q}_0(-k) - ik\tilde{g}_0(k^2) + ike^{ikL}\tilde{h}_0(k^2) + e^{k^2T}\hat{q}_T(-k)}{e^{-ikL} - e^{ikL}} \\
\widetilde{h}(k) &= \frac{2ik\widetilde{g}_0(k^2) - 2ike^{ikL}\widetilde{h}_0(k^2) - \hat{q}_0(k) + e^{k^2T}\hat{q}_T(k) + \hat{q}_0(-k) - e^{k^2T}\hat{q}_T(-k)}{e^{-ikL} - e^{ikL}}.
\end{aligned} \tag{2.19}$$

Finally we substitute $\widetilde{g}(k)$ and $\widetilde{h}(k)$ into the general solution (2.8):

$$\begin{aligned}
q(x,T) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\hat{q}_0(k)dk \cdots \\
&-\frac{1}{2\pi}\int_{\partial D^+} \frac{e^{ikx-k^2T}}{e^{-ikL} - e^{ikL}} \left[2ike^{-ikL}\widetilde{g}_0(k^2) - 2ik\widetilde{h}_0(k^2) - e^{ikL}\hat{q}_0(k) + e^{ikL+k^2T}\hat{q}_T(k) + e^{-ikL}\hat{q}_0(-k) - e^{-ikL+k^2T}\hat{q}_T(-k)\right] dk \\
&\quad -\frac{1}{2\pi}\int_{\partial D^-} \frac{e^{-ik(L-x)-k^2T}}{e^{-ikL} - e^{ikL}} \left[2ik\widetilde{g}_0(k^2) - 2ike^{ikL}\widetilde{h}_0(k^2) - \hat{q}_0(k) + e^{k^2T}\hat{q}_T(k) + \hat{q}_0(-k) - e^{k^2T}\hat{q}_T(-k)\right] dk.
\end{aligned}$$

In the Dirichlet case the $\hat{q}_T(\pm k)$ terms do not result in an extra contribution. This is because at $k = 0$ the numerator of the latter two integrands is also equal to zero, and thus $k = 0$ is not a pole but a *removable singularity*. That is, there should be analytic functions identical to our integrands at all other points in k which do not share this singularity. This means that the terms involving $\hat{q}_T(\pm k)$ can be considered analytic on the full regions D^+ and D^-, and they go to zero as $k \to \infty$, so we can use Cauchy's integral theorem with Jordan's lemma show the contour integrals of these terms is zero.

With these terms removed, we find the solution to the Dirichlet problem

$$\begin{aligned}
q(x,T) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx-k^2T}\hat{q}_0(k)dk \\
&-\frac{1}{2\pi}\int_{\partial D^+} \frac{e^{ikx-k^2T}}{e^{-ikL} - e^{ikL}} \left[2ike^{-ikL}\widetilde{g}_0(k^2) - 2ik\widetilde{h}_0(k^2) - e^{ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right] dk \\
&\quad -\frac{1}{2\pi}\int_{\partial D^-} \frac{e^{-ik(L-x)-k^2T}}{e^{-ikL} - e^{ikL}} \left[2ik\widetilde{g}_0(k^2) - 2ike^{ikL}\widetilde{h}_0(k^2) - \hat{q}_0(k) + \hat{q}_0(-k)\right] dk.
\end{aligned} \tag{2.20}$$

3

Improved version of the Kalimeris et al boundary value control algorithm

In their paper [10], Kalimeris et al. transform the classical boundary value problem to a control problem. In doing this, the question moves from "What will my final condition $q_T(x)$ be with these boundary values?" to "What boundary values do I need to reach a desired $q_T(x)$?". With this in mind, we will start referring to the boundary values as *(control) inputs* which lead to the output of the system, $q_T(x)$. We will also refer to the desired function of $q_T(x)$ as the *reference function*.

Kalimeris et al. [10] propose a new algorithm to numerically compute the control inputs. The algorithm approximates the input(s) as a sum of a set of basis functions. Then it considers the solution of the UTM in a number of points in x, given the initial condition $q_0(x)$. The contributions of each of the basis functions at each point in x can be calculated individually and they scale linearly with some constant. These contributions are then put in a matrix equation where they are compared to the desired output, this equation is then solved for the input(s).

We will describe the algorithm from Kalimeris et al. in Section 3.1. In Section 3.2 we will go over the proposed improvements. We will wrap up improvements in Section 3.3 and summarize our version of the algorithm.

3.1. The Kalimeris et al algorithm

We will start this section with the derivation of the algorithm from the solution of the heat equation, which was derived in the previous Chapter.

The examples in [10] were performed on the Neumann problem of the heat equation on the finite domain (2.1) with the reference function $q(x,T)=0$, and with the assumption that $g_1(t)=0$. Any initial conditions were chosen such that $\hat{q}_0(0)=0$, and the input $h_1(t)$ was 'restricted' such that $\tilde{h}_1(0)=0$[1]. This removes the effect of the constant contribution in (2.16). The solution (2.16) at time T then reduces to

$$\begin{aligned} q(x,T)=0=\frac{1}{2\pi}\int_{-\infty}^{\infty}e^{ikx-k^2T}\hat{q}_0(k)dk \\ -\frac{1}{2\pi}\int_{\partial D^+}\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[2\widetilde{h}_1(k^2)+e^{ikL}\hat{q}_0(k)+e^{-ikL}\hat{q}_0(-k)\right]dk \\ -\frac{1}{2\pi}\int_{\partial D^-}\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[2\widetilde{h}_1(k^2)+e^{-ikL}\hat{q}_0(k)+e^{-ikL}\hat{q}_0(-k)\right]dk. \end{aligned} \tag{3.1}$$

Kalimeris et al. also provide an additional way to simplify the computation of the integrals over ∂D^+ and ∂D^-. Namely, to deform the contours ∂D^+ and ∂D^- to C^+ and C^-, respectively, as shown

[1] In [10] this restriction is mentioned but it seems it is actually restricted is by choosing a $q_0(x)$ such that $\hat{q}_0(0)=0$. If the algorithm works then this is indeed a restriction, as if we examine the GR (2.3) at $k=0$ we find $\hat{q}_0(0)-\widetilde{g}_1(0)+\widetilde{h}_1(0)=\hat{q}_T(0)$. Including the other assumptions this means $\widetilde{h}_1(0)$ should indeed be zero.

in figure 3.1. The new contours are similar to ∂D^+ and ∂D^+ but they only form an angle of $\frac{\pi}{8}$ with the real axis. These angles should provide the integrals better exponential decay and thus improve computation.

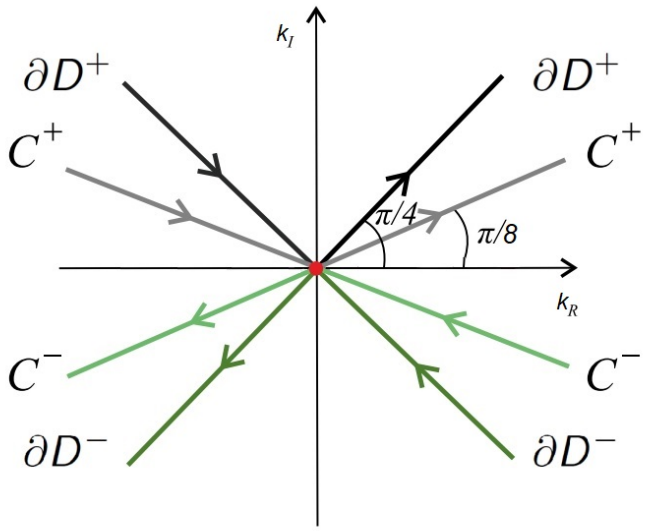


Figure 3.1: Integration contours $C^{\pm}$ compared to the contours $\partial D^{\pm}$.

(3.1) is then organised in such a way that all input-related integrals are on one side, and all other integrals (involving the initial condition and the wanted finals condition) are on the other side:

$$\begin{aligned}\frac{1}{\pi}\int_{C^+}\frac{e^{ikx}+e^{-ikx}}{e^{ikL}-e^{-ikL}}e^{-k^2T}\tilde{h}_1(k^2)dk = \frac{1}{2\pi}\int_{-\infty}^{\infty}e^{ikx-k^2T}\hat{q}_0(k)dk- \\ \frac{1}{2\pi}\int_{C^+}\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[e^{ikL}\hat{q}_0(k)+e^{-ikL}\hat{q}_0(-k)\right]dk \\ -\frac{1}{2\pi}\int_{C^-}\frac{e^{ikx-k^2T}}{e^{ikL}-e^{-ikL}}\left[e^{-ikL}\hat{q}_0(k)+e^{-ikL}\hat{q}_0(-k)\right]dk\end{aligned} \tag{3.2}$$

Here we combined the integrals involving $\tilde{h}_1(k^2)$ into the same integral using the transformation $k \to -k$ for the integral on ∂D^- (which would now be C^-).

The input is then approximated as a (as of yet unknown) linear combination of the first N basis functions of a basis. For the right Neumann boundary this would be

$$h_1(t) \approx \sum_{n=1}^{N}\alpha_n\phi_n(t), \tag{3.3}$$

where ϕ_n is the nth basis function. The function basis chosen by Kalimeris [10] is the sine function basis.

$$\phi_n(t) = \sin\left(\pi n\frac{t}{T}\right), \qquad t \in [0,T], \tag{3.4}$$

The T-transforms of these can be calculated algebraically, which will save some computation time later:

$$\int_0^T e^{k^2 s}\phi_n(s)ds = \tilde{\phi}_n(k^2) = \frac{-\pi nT(-1)^n e^{k^2T}+\pi nT}{k^4T^2+\pi^2n^2}. \tag{3.5}$$

The α_n of each function basis can be taken out of both the T-transform integral and the integral on

C^+ in (3.2):

$$\sum_{n=1}^{N} \frac{\alpha_n}{\pi} \int_{C^+} \frac{e^{ikx} + e^{-ikx}}{e^{ikL} - e^{-ikL}} e^{-k^2T} \tilde{\phi}_n(k^2) dk \approx \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ikx - k^2T} \hat{q}_0(k) dk - \frac{1}{2\pi} \int_{C^+} \frac{e^{ikx - k^2T}}{e^{ikL} - e^{-ikL}} \left[e^{ikL} \hat{q}_0(k) + e^{-ikL} \hat{q}_0(-k) \right] dk - \frac{1}{2\pi} \int_{C^-} \frac{e^{ikx - k^2T}}{e^{ikL} - e^{-ikL}} \left[e^{-ikL} \hat{q}_0(k) + e^{-ikL} \hat{q}_0(-k) \right] dk. \tag{3.6}$$

This equation is valid for all $x \in [0, L]$. This will be exploited by evaluating it in M points in x. We will call these points *sampling nodes*, and they are denoted as $\{x_m\}_{m=1}^{M}$ with $x_m \in [0, L]$. Evaluating (3.6) at these M nodes creates a system of M equations. The right hand side of these equations will be organised in a vector (with length M) that we will call Q. The elements of Q are given as

$$Q_m = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ikx_m - k^2T} \hat{q}_0(k) dk - \frac{1}{2\pi} \int_{C^+} \frac{e^{ikx_m - k^2T}}{e^{ikL} - e^{-ikL}} \left[e^{ikL} \hat{q}_0(k) + e^{-ikL} \hat{q}_0(-k) \right] dk - \frac{1}{2\pi} \int_{C^-} \frac{e^{ikx_m - k^2T}}{e^{ikL} - e^{-ikL}} \left[e^{-ikL} \hat{q}_0(k) + e^{-ikL} \hat{q}_0(-k) \right] dk \qquad m = 1, 2, .., M. \tag{3.7}$$

Next we introduce vector α which contains all α_n terms and factor this out of the left hand side of the system of equations. We denote the resulting matrix as F_h. We are left with the following system of equations:

$$F_h \alpha = Q, \tag{3.8}$$

where F_h is a $M \times N$ matrix with elements

$$F_{h;n,m} = \frac{1}{\pi} \int_{C^+} \frac{e^{ikx_m} + e^{-ikx_m}}{e^{ikL} - e^{-ikL}} e^{-k^2T} \tilde{\phi}_n(k^2) dk, \qquad n = 1, 2, .., N, \quad m = 1, 2, .., M, \tag{3.9}$$

and α a N length vector:

$$a = (a_1, a_2, \cdots a_N), \qquad n = 1, 2, .., N. \tag{3.10}$$

From here we proceed with the algorithm. For a fixed L, T, and a given $q_0(x)$, one should choose the number of basis functions N, the number of sampling nodes M, and the distribution of sampling nodes:

- Compute the values of matrices F_h and Q. These are found in (3.9) and (3.7), respectively.
- Solve for α using $\alpha = F^{-1}Q$.
- Use α in (3.3) to find the boundary values.

In [10], the algorithm is shown to work well under the chosen conditions mentioned earlier. The control inputs that are obtained from the algorithm are relatively small and the error, already in the range of 10^{-11}, is shown to decrease exponentially with an increasing N. However, it must be said that these results were found under quite specific conditions; Only nullcontrol[2] was performed, and only with initial conditions $q_0(x)$ that were chosen so that they would naturally decay to $q_T(x)$ anyway[3]. Besides this, we found that the robustness of the algorithm under small computational errors could be improved vastly. These issues, along with some other points of improvement, will be touched upon in the next section.

3.2. Additions to the algorithm

The following areas where the algorithm can be improved have been identified:

- Increased freedom of choice in the initial conditions and reference function.
- Implementation of control from both sides.

[2] Control with the intention of steering the system towards zero.

[3] $\hat{q}_0(0) = \int_0^L e^{i0x} q_0(x) dx = 0$ means the average of $q_0(x)$ is 0 so naturally this will tend to $q(x, T) \to 0$ for $T >> 0$.

- A more robust method of solving for α.
- Expansion to the Dirichlet problem.

Besides these, the choice of integration contours could also be further optimized. There is a number of contours of integration available in the literature [4], [5]. Compared to these, the contours $C^{\pm}$ are quite simple. However, they are far easier to implement and thus the choice has been made to continue to use C^+ and C^- going forward.

3.2.1. Arbitrary reference function and initial conditions

In Kalimeris et al, [10], initial conditions with $\hat{q}_0(0) = 0$ were chosen as they remove the impact of the constant contribution found in (2.16). Thus, to allow an arbitrary use of initial conditions we have to address this contribution. This is done simply by slightly indenting the contours of integration around $k = 0$ as described in Section 2.2 for the alternative solution of the Neumann problem (2.17). The new integration contours will be C_c^+ and C_c^-, which are similar to C^+ and C^-, but instead of passing through $k = 0$ they will go around it in a tiny circular (counterclockwise) arc.

We extend the method to use any control reference $q_T(x)$ (not just null-control) simply by adding the term $-q_T(x_m)$ to Q. The caveat here is that only M nodes from the reference function are sampled, so any information not specifically on these points is lost. This loss of information could lead to less accurate final result, as the final state can still deviate from the reference function at other points. We see two options to improve tracking performance without making fundamental changes to the algorithm.

1. Increase the number of sample nodes M. This should increase tracking performance. However, each added point in M not only means one extra integral to compute in Q, but also N extra integrals that need to be computed in F. This means that unless the extra precision is really needed, this is not seen as a desired option.
2. Improve the placement of the sampling nodes $\{x_m\}_{m=1}^M$. Strategic placement of the nodes should improve the tracking performance of the algorithm. For instance, placing $\{x_m\}_{m=1}^M$ on both the minima and maxima of a sinusoidal q_T should convey nearly all the important characteristics of the function to the algorithm, which should improve tracking of the reference function. We note that the viability of this approach will probably be heavily dependant on the reference function at hand. We will further elaborate on the exact placements of these points in chapter 5.

With these changes, (3.9) and (3.7) now become

$$F_{h;n,m} = \frac{1}{\pi}\int_{C_c^+} \frac{e^{ikx_m} + e^{-ikx_m}}{e^{ikL} - e^{-ikL}} e^{-k^2T}\tilde{\phi}_n(k^2)dk, \qquad n = 1, 2, .., N, \quad m = 1, 2, .., M, \tag{3.11}$$

$$\begin{aligned} Q_m = -q(x_m, T) + \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx_m - k^2T}\hat{q}_0(k)dk - \frac{1}{2\pi}\int_{C_c^+} \frac{e^{ikx_m - k^2T}}{e^{ikL} - e^{-ikL}}\left[e^{ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk \\ -\frac{1}{2\pi}\int_{C_c^-} \frac{e^{ikx_m - k^2T}}{e^{ikL} - e^{-ikL}}\left[e^{-ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk \qquad m = 1, 2, .., M. \end{aligned} \tag{3.12}$$

3.2.2. Control from both sides

We achieve control from both sides by performing the derivation of the algorithm again, but without setting $g_1(t) = 0$. Instead we use

$$g_1(t) = \sum_{n=1}^{N} \beta_n \phi_n(t). \tag{3.13}$$

At the point where we would factor out vector α and the matrix F_h, we now factor out vector $\gamma = (\beta, \alpha)^T$ out of matrix $F = (F_g, F_h)$. Here, β and F_g fulfill the same purpose as α and F_h, but for the boundary value $g_1(t)$; F_g and F_h are both $M \times N$ matrices, and α and β are N length vectors. Each element of F_g is given by

$$F_{g;n,m} = -\frac{1}{\pi}\int_{C_c^+} \frac{e^{ik(L-x_m)} + e^{-ik(L-x_m)}}{e^{ikL} - e^{-ikL}} e^{-k^2T}\tilde{\phi}_n(k^2)dk, \qquad n = 1, 2, .., N, \quad m = 0, 1, .., M. \tag{3.14}$$

Instead of solving (3.8), we now solve

$$F\gamma = Q, \tag{3.15}$$

and the boundary values $h_1(t)$ and $g_1(t)$ can be found by (3.3) and by (3.13), respectively.

3.2.3. Method of solving for gamma

The final step of the algorithm involves solving for the vector γ (or α, in the case of control from just the right side). In [10] this is done by simply done by the inverse of F, with $F^{-1}Q = \gamma$. This will be problematic when when the matrix F near-singular, which we have found to often be the case for the numerical examples in this work. A near-singular F means that a small perturbation in a part of F_h (of say a discretization or rounding error) will lead to a much larger change in F^{-1} and thereby, in γ. This could lead to values found for γ that are orders of magnitude greater than needed for the problem at hand. While in theory this might still lead to a well-controlled system (up to the small perturbation), temperature input values of 10^6 will be less nice in practise.

Therefore, in addition to minimizing the (squared) error at the sampling nodes

$$\min_{\gamma} \|F\gamma - Q\|_2^2, \tag{3.16}$$

we also want to implement some measure to keep the boundary values relatively small. For this we introduce a weight on the input in our optimization process in the form $\gamma^T W \gamma$, where W is a $2N \times 2N$ weighting matrix.

To actually add the weights to the optimization problem we start writing out (3.16):

$$\begin{aligned}\min_{\gamma} \|F\cdot\gamma - Q\|_2^2 &= \min_{\gamma}(F\cdot\gamma - Q)^T(F\cdot\gamma - Q)\\ &= \min_{\gamma}\gamma^T F^T F\gamma - \gamma^T F^T Q - Q^T F\gamma + Q^T Q \quad = \min_{\gamma}\gamma^T F^T F\gamma - 2\gamma^T F^T Q + Q^T Q,\end{aligned}$$

where we see we can simply add our weights to the first term. The term $Q^T Q$ is independent of γ, and can therefore be removed.

$$\min_{\gamma}\gamma^T(F^T F + W)\gamma - 2\gamma^T F^T Q \tag{3.17}$$

This is a standard quadratic programming problem, which has the solution $\gamma = (F^T F + W)^{-1}F^T Q$ [12]. Should it be wanted, this form also allows constraints to be put on γ, such as upper and lower bounds.

How W should be chosen (and possibly tuned) will depend on its purpose. If one would want to limit rapidly changing boundary values one might put a higher weighting term on parts of γ that correspond to a higher N. In our case we just want to limit the total control output, $\int_0^T [|g_1(t)|^2 + |h_1(t)|^2]dt$. Since our basis functions are orthogonal

$$\int_0^T \sin\left(\pi n\frac{t}{T}\right)\sin\left(\pi m\frac{t}{T}\right)dt = 0 \qquad \forall n \neq m,$$

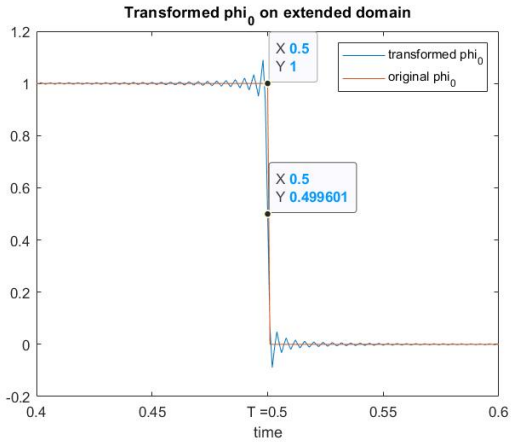
and the total output of each basis function is identical

$$\int_0^T \sin\left(\pi n\frac{t}{T}\right)^2 dt = T/2 \qquad \forall n,$$

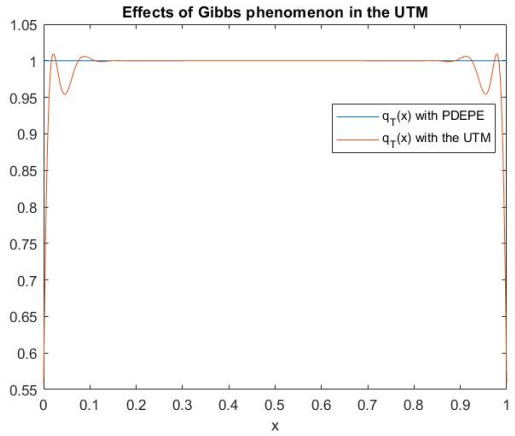
our choice of W will be a diagonal matrix with identical weighting terms w.

3.2.4. Expansion of basis for Dirichlet control

To implement the control algorithm for the Dirichlet problem, we start with its solution of the UTM on the finite domain (2.20). From there we continue in the same way as we did for the Neumann case; Organize the solution into input and non-input related terms, define the input(s) as a linear combination of basis functions, and write it as a matrix equation. But before going further with this, there is another problem to look into, which concerns the control of the points $q_T(0)$ and $q_T(L)$.



(a) Visualisation of the Gibbs phenomenon. Signal $\phi_0(t)$ is transformed and transformed back and is deformed on the edge of its domain.



(b) Effects of the Gibbs phenomenon on the UTM. Comparison between the UTM and Matlab's PDEPE with $g_0(t) = h_0(t) = q_0(x) = 1$.

Figure 3.2: Gibbs phenomenon effects. The results shown in these figures were computed using the values in table 5.1 in Chapter 5.

So far we have been using sums of sine waves in (3.3) and (3.13) to approximate our inputs. Each of these sine waves, and therefore also their sum, has a value of zero at $x = 0$ and $x = L$ for $t = T$. From this one can immediately see that it is impossible to correctly represent any nonzero point $q_T(0)$ or $q_T(L)$ when using the same basis for Dirichlet control (as by definition $q_T(0) = g_0(T)$ and $q_T(L) = h_0(T)$). With this in mind, we are forced to extend the basis to

$$\phi_n(t) = \begin{cases} \sin\left(\pi n \frac{t}{T}\right), & n = 1, 2, .., N, \quad t \in [0, T], \\ 1, & n = 0, \quad t \in [0, T], \end{cases} \tag{3.18}$$

We approximate the boundary values by

$$g_0(t) = \sum_{n=0}^{N} \beta_n \phi_n(t), \qquad h_0(t) = \sum_{n=0}^{N} \alpha_n \phi_n(t). \tag{3.19}$$

Finally we note that the T-transform of $\phi_0(t)$ can also be computed in advance:

$$\int_0^T e^{k^2 s} \phi_0(s) ds = \widetilde{\phi}_0(k^2) = \frac{e^{k^2 T} - 1}{k^2}.$$

Unfortunately this solution brings other problems with it, as the sharp cutoff of $\phi_0(t)$ causes some artifacts one its edges once transformed during numerical calculation (the sharp cutoff here would be the drop-off from 1 to 0 for $t \notin [0, T]$). These edge artifacts are caused by the *Gibbs phenomenon*, an effect that occurs when a Fourier-type transform is performed with a finite summation [11](in our case this is the truncation and discretization of the integrals over $\partial D^{\pm}$). The effects can be seen in figure 3.2a.

In figure 3.2a the signal $\phi_0(t)$ and its transformed version (T-transform and back) are plotted near the edge of the signal at T. Along with some other artifacts near T, one can see that at $t = T$ the transformed ϕ_0 has only half the value of the original ϕ_0. This creates significant problems, particularly in the Dirichlet problem. We can see this in figure 3.2b, where we compare the UTM's and Matlabs PDEPE solutions for the Dirichlet problem with $g_0(t) = h_0(t) = q_0(x) = 1$. The solution found by PDEPE is $q_T(x) = 1$ (by common sense, this is also what the solution should be), but the solution found by the (discrete implementation of) UTM is influenced by the artifacts created by the Gibbs phenomenon, which leave the values on the edges about half of what they should be.

To avoid most problems caused by these artifacts we simply obtain the values of α_0 and β_0 by setting them to $q_T(L)$ and $q_T(0)$, respectively. This is possible since we already know our desired $q_T(x)$ and

because by definition $q_T(0) = g_0(T)$ and $q_T(L) = h_0(T)$ (and because all other basis functions have a value of 0 at $x = 0, L$). Since these values are known, they will be separated from the rest of the integrals concerning the boundary equation and moved to the right side of the equation we're solving, (3.15), and moved into vector Q. With this, the elements of F and Q for the Dirichlet case are

$$F_{g;n,m} = \frac{1}{\pi}\int_{C^+} \frac{e^{ik(x_m-L)} - e^{-ik(x_m-L)}}{e^{-ikL} - e^{ikL}} ike^{-k^2T}\widetilde{\phi}_n(k^2)dk, \qquad n = 1, 2, .., N, \quad m = 1, 2, .., M, \tag{3.20}$$

$$F_{h;n,m} = \frac{1}{\pi}\int_{C^+} \frac{e^{ikx_m} - e^{-ikx_m}}{e^{-ikL} - e^{ikL}} ike^{-k^2T}\widetilde{\phi}_n(k^2)dk, \qquad n = 1, 2, .., N, \quad m = 1, 2, .., M, \tag{3.21}$$

and

$$\begin{aligned} Q_m = -q(x_m, T) + \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx_m - k^2T}\hat{q}_0(k)dk - \frac{1}{2\pi}\int_{C^+} \frac{e^{ikx_m-k^2T}}{e^{-ikL} - e^{ikL}}\left[e^{-ikL}\hat{q}_0(-k) - e^{ikL}\hat{q}_0(k)\right]dk \\ -\frac{1}{2\pi}\int_{C^-} \frac{e^{ikx_m-k^2T}}{e^{-ikL} - e^{ikL}}\left[e^{-ikL}\hat{q}_0(-k) - e^{-ikL}\hat{q}_0(k)\right]dk - \frac{\beta_0}{\pi}\int_{C^+} \frac{e^{ik(x_m-L)} - e^{-ik(x_m-L)}}{e^{-ikL} - e^{ikL}} ike^{-k^2T}\widetilde{\phi}_0(k^2)dk \\ -\frac{\alpha_0}{\pi}\int_{C^+} \frac{e^{ikx_m} - e^{-ikx_m}}{e^{-ikL} - e^{ikL}} ike^{-k^2T}\widetilde{\phi}_0(k^2)dk, \qquad m = 1, 2, .., M. \end{aligned} \tag{3.22}$$

Additionally we add the requirement that any integration node x_m may not be placed too close to the edges of the domain of $x \in [0, L]$. The artifacts caused by the Gibbs phenomenon are still present in the last two integrals of Q, and an x_m that is placed to close to either 0 or L might lead to an error in the corresponding element of Q, which will consequently affect the entire solving process. The limitation of not being able to place nodes close to the edges is balanced by exactly knowing the values of α_0 and β_0, which effectively serve as two sampling nodes that don't need to be included in the solving process.

For the Neumann case we apply the same expansion of basis and addition to the vector Q. As we're dealing with the derivative (in x) of $q(x, t)$ and not the function value, the inclusion of ϕ_0 isn't as needed as it is in the Dirichlet case, but it is still useful in more accurately representing certain functions.

The functions $g_1(t)$ and $f_1(t)$ will be approximated by

$$g_1(t) = \sum_{n=0}^{N} \beta_n\phi_n(t), \qquad h_1(t) = \sum_{n=0}^{N} \alpha_n\phi_n(t), \tag{3.23}$$

and the elements of Q will be

$$\begin{aligned} Q_m = -q(x_m, T) + \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx_m - k^2T}\hat{q}_0(k)dk - \frac{1}{2\pi}\int_{C_c^+} \frac{e^{ikx_m-k^2T}}{e^{ikL} - e^{-ikL}}\left[e^{ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk \\ -\frac{1}{2\pi}\int_{C_c^-} \frac{e^{ikx_m-k^2T}}{e^{ikL} - e^{-ikL}}\left[e^{-ikL}\hat{q}_0(k) + e^{-ikL}\hat{q}_0(-k)\right]dk + \frac{\beta_0}{\pi}\int_{C_c^+} \frac{e^{ik(L-x_m)} + e^{-ik(L-x_m)}}{e^{ikL} - e^{-ikL}} e^{-k^2T}\tilde{\phi}_0(k^2)dk \\ -\frac{\alpha_0}{\pi}\int_{C_c^+} \frac{e^{ikx_m} + e^{-ikx_m}}{e^{ikL} - e^{-ikL}} e^{-k^2T}\tilde{\phi}_0(k^2)dk \qquad m = 1, 2, .., M, \end{aligned} \tag{3.24}$$

where we find α_0 and β_0 from our desired $q_T(x)$ as $\alpha_0 = \frac{d}{dx}q_T(L)$ and $\beta_0 = \frac{d}{dx}q_T(0)$. The F_g and F_h matrices will remain the same.

We also note that the unwanted artifacts created by the Gibbs phenomenon are less defined in the Neumann case (as only the derivative, and not the function value will be deformed). This means we can be more relaxed on the limitation of not placing tracking nodes x_m too close to the edges, though we should still be careful.

3.3. Final algorithm

With all the changes in place we now present the final algorithm. It can be used both for the Dirichlet and for the Neumann problem. For any fixed L, T, a given $q_0(x)$ and a desired $q_T(x)$[4], one should choose

[4]Technically, we don't need the entire function $q_T(x)$ as the algorithm only requires $q_T(x_m)$, $q_T(0)$, and $q_T(L)$. In theory this could be done with a set of data points, too

the amount of basis functions N, the amount of sampling nodes M, the distribution of the sampling nodes $\{x_m\}_{m=0}^{M}$, and a $2N \times 2N$ weighting matrix W. These sampling nodes should all be placed a 'safe' distance away from the edges of the domain of x.

- Evaluate $q(x, T)$ to find α_0 and β_0.
- Compute matrices $F = [F_g, F_h]$ and Q. For the Dirichlet problem, the elements of these matrices are given by (3.20), (3.21), and (3.22) respectively. For the Neumann problem the elements are given by (3.14), (3.11) and (3.24), respectively.
- Solve the optimization problem (3.17) to find $\gamma = [\beta; \alpha]$.
- Combine γ with α_0 and β_0 to find the boundary values from (3.23) for the Neumann problem or from (3.19) for the Dirichlet problem.

4

Boundary value control algorithm in the complex plane

One of the main drawbacks of the augmented Kalimeris et al. algorithm described in Section 3.3 is the relatively high computational cost per function evaluation point, $N+1$ integrals that need to be computed per x_m (or $2N+1$ when controlling both boundary values). In this chapter we will try to solve the same problem, but without the need to compute all these integrals. To do this, we have to go back quite deep into the derivation of the UTM, to the point before we apply the inverse Fourier transform to the GR:

$$\hat{q}_0(k) - ik\widetilde{g}_0(k^2) - \widetilde{g}_1(k^2) + ike^{-ikL}\widetilde{h}_0(k^2) + e^{-ikL}\widetilde{h}_1(k^2) = e^{k^2T}\hat{q}_T(k), \qquad k \in \mathbb{C}. \tag{4.1}$$

This equation, like any solution to the heat equation, (2.17) or (2.20), relates functions we want to know, the boundary values, to functions we know or can specify, the initial condition and $q(x,T)$. Thus, theoretically, we should be able to apply the same approach as in the previous chapter: Evaluate this function at M points and approximate the boundary values as sums of N basis functions, then write this as a matrix equation and solve it for the boundary values. The big benefit of carrying out this process on the GR (as opposed to carrying it out on the full solution (2.16)), is that setting up the matrix equation does not involve the computation of any integrals (beside the Fourier transforms of the initial data), so we should be able to increase both N and M by quite a bit while keeping the overall computation time low.

There are, however, some difficulties:

- Since the GR still contains all available boundary values (and not only the Dirichlet/Neumann ones), we need to solve for all the boundary values instead of solving for just two of them. This will not make the problem any more complex, but it will increase the scale of the problem.
- Instead of approximating a function on $[0,L] \in \mathbb{R}$, we are approximating a function on the entire $\mathbb{C}$ plane.
- The functions we find for the boundary values are found in the complex plane, and any errors in them may be enhanced when they are transformed back into the real domain.

These are some valid concerns and it remains to be seen if the lower computational cost per point evaluation can make up for it.

We will derive the algorithm is Section 4.1. In the derivation we will also discuss the optimal version(s) of the GR and the solving method for the boundary values. A final summary of the algorithm will be given in Section 4.2.

4.1. Derivation of the second algorithm

As stated earlier, somewhere in the derivation of this algorithm we plan to evaluate some version of the GR at M nodes $\{k_m\}_{m=1}^M$. Before doing any other parts of the derivation, we will examine the GR

to gain some insight on the optimal placement of these nodes.

We start by observing that although the GR (4.1) is valid on the whole k-plane, there are some regions where it's unbounded as $k \to \infty$. Any evaluating of the GR in these regions will be orders of magnitude larger than any evaluations in bounded regions, and will massively outweigh them in an error-based solver. For this reason we will constrain ourselves to regions in which any exponentials are non-increasing, and the integral transforms are bounded for $k \to \infty$. For (4.1) this is the region D^-, as defined in (2.5a), for both exponentials e^{-ikL} and e^{k^2T} are bounded there.

Next we will point out the focus of the GR. We notice that as we increase the magnitude of k, that some terms decay faster than others. If we consider the terms $-\tilde{g}_1(k^2)$ and $e^{ikL}\tilde{h}_1(k^2)$ in (4.1) on the contour ∂D^- (a contour on which all exponentials are bounded), we can see that $e^{ikL}\tilde{h}_1(k^2)$ decays $e^{|k|L/\sqrt{2}}$ times faster than $\tilde{g}_1(k^2)$ as we let $|k| \to \infty$ (if we assume equal $\tilde{h}_1(k^2)$ and $\tilde{g}_1(k^2)$). This means that when we use the least squares solving approach later, any errors in $\tilde{h}_1(k^2)$ will be less accounted for. We would say the emphasis of this GR is on $\tilde{g}_1(k^2)$, as its coefficients decay less as $k \to \infty$. We want the algorithm to evaluate any errors in $\tilde{h}(k)$ and $\tilde{h}(k)$ evenly, and for this we will use the following two equations

$$e^{-k^2T}\left[-ik\widetilde{g}_0(k^2) - \widetilde{g}_1(k^2) + ike^{-ikL}\widetilde{h}_0(k^2) + e^{-ikL}\widetilde{h}_1(k^2)\right] = \hat{q}_T(k) - e^{-k^2T}\hat{q}_0(k), \qquad k \in E^-, \tag{4.2a}$$

$$e^{-k^2T}\left[-ike^{ikL}\widetilde{g}_0(k^2) - e^{ikL}\widetilde{g}_1(k^2) + ik\widetilde{h}_0(k^2) + \widetilde{h}_1(k^2)\right] = e^{ikL}\hat{q}_T(k) - e^{-k^2T+ikL}\hat{q}_0(k), \qquad k \in E^+. \tag{4.2b}$$

The regions in which everything is bounded is region E^- for (4.2a) and region E^+ for (4.2b) (these regions were introduced in (2.5a)). In those respective regions both equations emphasize $\hat{q}_T(k)$ over $\hat{q}_0(k)$. The first equation emphasizes $\widetilde{g}_0$ and $\widetilde{g}_1$ more while in (4.2b) emphasizes $\widetilde{h}_0$ and $\widetilde{h}_1$ more. For our $\{k_m\}_{m=0}^M$ selection, if we want to emphasize both $\hat{q}_T(k)$ and all the boundary values, we need to select nodes for both equation in their respective bounded regions.

4.1.1. Derivation of $\hat{F}\gamma = \hat{G}$

We will perform our derivation on (4.2a). The derivation will be the same as for (4.2b), so when we're done we can simply multiply results by e^{ikL} to find its equivalent solution. We start with isolating the boundary values from the other terms of (4.2a) and approximating the boundary values as a linear combination of basis functions:

$$e^{-k^2T}\left[-ik\widetilde{g}_0(k^2) - \widetilde{g}_1(k^2) + ike^{-ikL}\widetilde{h}_0(k^2) + e^{-ikL}\widetilde{h}_1(k^2)\right] = \hat{q}_T(k) - e^{-k^2T}\hat{q}_0(k), \qquad k \in E^-. \tag{4.3}$$

$$g_0(t) = \sum_{n=0}^{N} \beta_{0;n}\phi_n(t), \qquad g_1(t) = \sum_{n=0}^{N} \beta_{1;n}\phi_n(t), \tag{4.4a}$$

$$h_0(t) = \sum_{n=0}^{N} \alpha_{0;n}\phi_n(t), \qquad h_1(t) = \sum_{n=0}^{N} \alpha_{1;n}\phi_n(t), \tag{4.4b}$$

with

$$\phi_n(t) = \begin{cases} \sin\left(\pi n \dfrac{t}{T}\right), & n = 1, 2, .., N, \quad t \in [0, T], \\ 1, & n = 0, \quad t \in [0, T], \end{cases} \tag{4.5}$$

As described in Section 3.2.4, we can directly obtain coefficients $\beta_{0;0}$, $\beta_{1;0}$, $\alpha_{0;0}$, and $\alpha_{1;0}$ from our q_T. As the coefficients for them are now known, we split off the ϕ_0 bases from the summations and

move them to the right side of the equation:

$$e^{-k^2T}\left[-ik\sum_{n=1}^{N}\beta_{0;n}\widetilde{\phi}_n(k^2)-\sum_{n=1}^{N}\beta_{1;n}\widetilde{\phi}_n(k^2)+ike^{-ikL}\sum_{n=1}^{N}\alpha_{0;n}\widetilde{\phi}_n(k^2)+e^{-ikL}\sum_{n=1}^{N}\alpha_{1;n}\widetilde{\phi}_n(k^2)\right]= e^{-k^2T}\left(ik\beta_{0;0}+\beta_{1;0}-ike^{-ikL}\alpha_{0;0}-e^{-ikL}\alpha_{1;0}\right)\widetilde{\phi}_0(k^2)+\hat{q}_T(k)-e^{-k^2T}\hat{q}_0(k),\qquad k\in E^-. \tag{4.6}$$

We want to evaluate this equation at M points in $k\in E^-$, which will be our nodes $\{k_m\}_{m=1}^M$. This will create a system of M equations. Next we factor out all α and β terms from the left hand side of system of equations, creating the matrix equation

$$\hat{F}_-\gamma=\hat{Q}_-, \tag{4.7}$$

where $\hat{F}_-$ is a $M\times 4N$ matrix, γ is a $4N$ length vector, and $\hat{Q}_-$ is a M length vector. As this matrix equation is in the complex plane, a hat symbol is added to both $\hat{F}_-$ and $\hat{Q}_-$ to distinguish them from their counterparts in the Kalimeris et al. algorithm. The matrix $\hat{F}_-$ is given by

$$\hat{F}_-=\begin{bmatrix} -ik_1e^{-k_1^2T}\widetilde{\Phi}(k_1^2) & -e^{-k_1^2T}\widetilde{\Phi}(k_1^2) & ik_1e^{-k_1^2T-ik_1L}\widetilde{\Phi}(k_1^2) & e^{-k_1^2T-ik_1L}\widetilde{\Phi}(k_1^2)\\ -ik_2e^{-k_2^2T}\widetilde{\Phi}(k_2^2) & -e^{-k_2^2T}\widetilde{\Phi}(k_2^2) & ik_2e^{-k_2^2T-ik_2L}\widetilde{\Phi}(k_2^2) & e^{-k_2^2T-ik_2L}\widetilde{\Phi}(k_2^2)\\ \vdots & \vdots & \vdots & \vdots\\ -ik_Me^{-k_M^2T}\widetilde{\Phi}(k_M^2) & -e^{-k_M^2T}\widetilde{\Phi}(k_M^2) & ik_Me^{-k_M^2T-ik_ML}\widetilde{\Phi}(k_M^2) & e^{-k_M^2T-ik_ML}\widetilde{\Phi}(k_M^2)\end{bmatrix}, \tag{4.8}$$

where $\widetilde{\Phi}(k_m^2)$ is a $1\times N$ vector given by

$$\widetilde{\Phi}(k_m^2)=\begin{bmatrix}\widetilde{\phi}_1(k_m^2) & \widetilde{\phi}_2(k_m^2) & \cdots & \widetilde{\phi}_n(k_m^2)\end{bmatrix}$$

The elements of the vector $\hat{Q}_-$ are

$$\hat{Q}_{-;m}=e^{-k_m^2T}\left(ik_m\beta_{0;0}+\beta_{1;0}-ik_me^{-ik_mL}\alpha_{0;0}-e^{-ik_mL}\alpha_{1;0}\right)\widetilde{\phi}_0(k_m^2)+\hat{q}_T(k_m)-e^{-k_m^2T}\hat{q}_0(k_m). \tag{4.9}$$

Finally, γ is given as

$$\gamma=\begin{bmatrix}\beta_{0;1} & \beta_{0;2} & \cdots & \beta_{0;n} & \beta_{1;1} & \beta_{1;2} & \cdots & \beta_{1;n} & \alpha_{0;1} & \alpha_{0;2} & \cdots & \alpha_{0;n} & \alpha_{1;1} & \alpha_{1;2} & \cdots & \alpha_{1;n}\end{bmatrix}^T.$$

The same derivation is done for (4.2b), with a new set of M nodes. These nodes are only placed in E^+. This results in the matrix equation

$$\hat{F}_+\gamma=\hat{Q}_+. \tag{4.10}$$

Here $\hat{F}_+$ and $\hat{Q}_+$ are the same sizes as $\hat{F}_-$ and $\hat{Q}_-$, respectively, and they are related by

$$\hat{F}_+=e^{ikL}\hat{F}_-,\qquad \hat{Q}_+=e^{ikL}\hat{Q}_-. \tag{4.11}$$

Finally, we combine the two matrix equations. The resulting matrix equation is derived from both equations (4.2a) and (4.2b), thus balancing the emphasis between the sets of boundary conditions from both sides.

$$\hat{F}\gamma=\hat{Q}=\begin{bmatrix}\hat{F}_-\\ \hat{F}_+\end{bmatrix}\gamma=\begin{bmatrix}\hat{Q}_-\\ \hat{Q}_+\end{bmatrix}. \tag{4.12}$$

4.1.2. Solving method for γ

We wish to solve for γ in the same way as we do for the Kalimeris et al. algorithm, by writing the least squared error problem as a quadratic programming problem and adding weights. However, since we are dealing with complex valued vectors and matrices in $\hat{Q}$ and $\hat{F}$, we will need to take some extra steps. We start by minimizing the squared error

$$\min_{\gamma}\|\hat{F}\gamma-\hat{Q}\|_2^2, \tag{4.13}$$

Here we notice that since γ is real, we can write this as

$$\begin{aligned}\min_{\gamma} \|\hat{F}\gamma - \hat{Q}\|_2^2 &= \min_{\gamma} \|\mathrm{Re}\{\hat{F}\}\gamma - \mathrm{Re}\{\hat{Q}\} + i\mathrm{Im}\{\hat{F}\}\gamma - i\mathrm{Im}\{\hat{Q}\}\|_2^2 \\ &= \min_{\gamma} \left\| \begin{bmatrix}\mathrm{Re}\{\hat{F}\}\\ \mathrm{Im}\{\hat{F}\}\end{bmatrix}\gamma - \begin{bmatrix}\mathrm{Re}\{\hat{Q}\}\\ \mathrm{Im}\{\hat{Q}\}\end{bmatrix}\right\|_2^2.\end{aligned}$$

The problem is now written in a form where all the matrices involved only contain real values. Writing this problem out and adding a $4N \times 4N$ weight matrix $\hat{W}$, we find the following quadratic programming problem which we can solve to find γ.

$$\min_{\gamma} \gamma^T \left(\begin{bmatrix}\mathrm{Re}\{\hat{F}\} & \mathrm{Im}\{\hat{F}\}\end{bmatrix}^T \begin{bmatrix}\mathrm{Re}\{\hat{F}\}\\ \mathrm{Im}\{\hat{F}\}\end{bmatrix} + \hat{W}\right)\gamma - 2\gamma^T \begin{bmatrix}\mathrm{Re}\{\hat{F}\} & \mathrm{Im}\{\hat{F}\}\end{bmatrix}^T \begin{bmatrix}\mathrm{Re}\{\hat{Q}\}\\ \mathrm{Im}\{\hat{Q}\}\end{bmatrix} \tag{4.14}$$

This can then be solved as any noncomplex valued quadratic programming problem.

4.2. Final k-plane algorithm

We will now describe the finished k-plane algorithm. It is usable for both the Dirichlet, and the Neumann problem. For a fixed L and T, a given $q_0(x)$, and a desired $q_T(x)$, one should choose the number of basis functions used N, (half) the number of sampling nodes M, and a weighting matrix W. One should also choose distributions of $\{k_m\}_{m=1}^M$ in both the regions E^- and E^+ (any distribution chosen in one region can simply be mirrored). From there:

- Evaluate $q(x,T)$ to find $\beta_{0;0}$, $\beta_{1;0}$, $\alpha_{0;0}$, and $\alpha_{1;0}$.
- Compute matrices $\hat{F}_-$, $\hat{Q}_-$, $\hat{F}_+$, and $\hat{Q}_+$. These are given by (4.8), (4.9), and (4.11), respectively.
- Solve the optimization problem given by (4.14) to find γ.
- Use the appropriate values from γ combined with the corresponding $\beta_{0;0}$ and $\alpha_{0;0}$, or $\beta_{1;0}$ and $\alpha_{1;0}$ to find the boundary values for the problem at hand, Dirichlet or Neumann. These are found using (4.4a)

5

Methodology and results

This Chapter is dedicated to the the testing and comparison of the two algorithms. This will be done by evaluating them in three different cases, each of which with a different desired $q_T(x)$ to test the operational range of the algorithms. The testing of the algorithms will be done for both Neumann and Dirichlet problems. If needed, the weight and the model parameters N and M, or $\hat{N}$ and $\hat{M}$, will be tuned. Once the boundary values are found, we will use them to calculate the solution to the UTM and to Matlabs PDEPE solver [7], which we will denote these solutions as $\overline{q_{T,\text{utm}}}(x)$ and $\overline{q_{T,\text{pdepe}}}(x)$, respectively. For verification we will only use the solution found by Matlab's PDEPE solver. We will use the the mean squared error (MSE) between $\overline{q_{T,\text{pdepe}}}(x)$ and our reference function q_T as our main verification metric. It will be calculated as

$$\text{MSE}(q_T(x), \overline{q_T}(x)) := \frac{\int_0^L [q_T(x) - \overline{q_{T,\text{pdepe}}}(x)]dx}{L}. \tag{5.1}$$

We'll exclude the solution found by the UTM in our verification process as this solution suffers from significant artifacts from the Gibbs phenomenon, which will heavily impact any verification method. Besides this it is generally good practise to have a separated verification method; Verification of a method based on the UTM with the UTM might get some biased results. For the sake of completeness the solution found by the UTM will still be displayed in figures. This way we can still visually confirm the UTM still performs well were it not for these Gibbs phenomenon artifacts. Finally we will touch on the control input, which will be measured by its L_2 norm, and the computation time.

5.1. Methodology for both algorithms

The computations required for the algorithm and the verification thereof were implemented in Matlab. Any continuous variables were discretized (with uniform step sizes). The step sizes for this can be seen in table 5.1. Any integral with k going to $\pm\infty$ will be truncated at $|k| = K_{\text{cutoff}}$, which can also be seen in the table. The circular arc in used in contours C_c^+ and C_c^- was approximated with radius k_{step}, and with c points along its arc, where c can also be found in the table. Note that all of these values are not optimized for speed, but were instead chosen small/big enough so that the code runs smoothly while still keeping the computation time practical.

Name	Value	Description
x_{step}	0.005	step size in x
t_{step}	0.005	step size in t
k_{step}	0.01	step size in k
K_{cutoff}	100	magnitude of the cutoff point of integrals of k going to ∞
c	31	number of points used to approximate the circular arc in the contours C_c^+ and C_c^-
w	10^{-7}	weight used in weighting matrix W

Name	Value	Description
x_{edge}	0.15	distance from 0, L we can't pick x_m in.
$\hat{K}_{cutoff}$	40	The maximum value used for picking k_m.

Table 5.1: Table of used values

In all of these tests, we will use $T = 0.5$ and $L = 1$. When using the Kalimeris algorithm, we will not pick any x_m closer than x_{edge} to the edges $x = 0$, $x = L$. This is done to avoid the effects of the earlier mentioned Gibbs phenomenon. This phenomenon heavily depends on K_{cutoff}, and for its current value empirical testing has established that there should not be any artifacts for $x_{\text{edge}} = 0.15$ for the Dirichlet problem. This value could be made smaller for the Neumann problem, but it was kept the same for simplicity.

5.1.1. Kalimeris algorithm x_m placement

For the Kalimeris algorithm, we propose three possible options for the placement of sampling nodes $\{x_m\}_1^M$. The first is simply the uniform distribution, with the points are spread equidistant over the available domain $[x_{\text{edge}}, L - x_{\text{edge}}]$. For our second option we turn to [10], where Kalimeris et al. reported a lower error when using a distribution of $\{x_m\}_1^M$ which was denser near their control input, $h_1(t)$. We plan to use the same for our situation; A $\{x_m\}_1^M$ distribution which is denser near the edges and more sparse in the middle of the x domain (while still conforming to our x_{edge} limitations). We will use the following formula for our distribution:

$$x_m = \frac{L}{2} + \frac{L - 2x_{\text{edge}}}{2} \cos\left(\frac{2k-1}{2K}\pi\right), \qquad k = 1, 2, .., K, \tag{5.2}$$

which is is the formula for *Chebyshev nodes* on $[x_{\text{edge}}, L - x_{\text{edge}}]$ [15]. The distribution of Chebyshev nodes for $M = 10$ can be seen in figure 5.1. Chebyshev nodes are normally used to minimize error for polynomial interpolation, and it will be tested if the property of error minimisation holds for our algorithm.

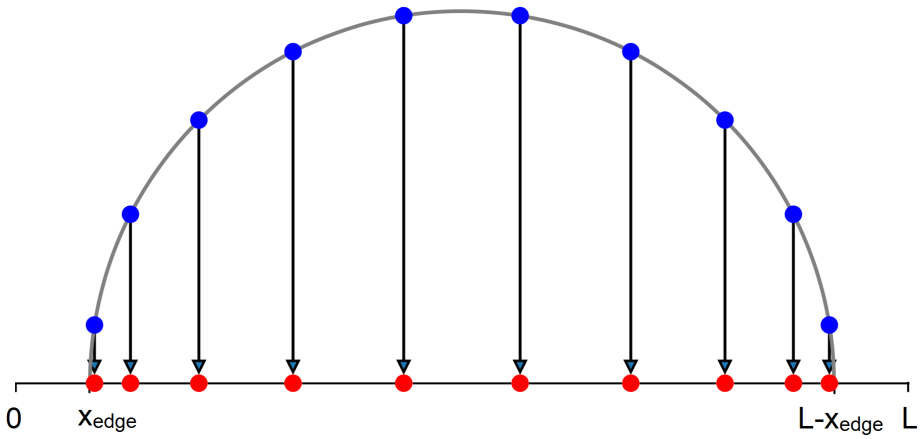


Figure 5.1: Chebyshev nodes for $M = 10$ on the $[x_{\text{edge}}, L - x_{\text{edge}}]$ interval. The nodes are placed equidistant on a semicircle and then projected on the real line. Modified version of [9]

The third option of x_m placement will to just pick the location of the nodes ourselves. For this option the placement will rely on q_T, and it will allow us to accommodate any irregularities in the reference function. We also note that in theory x_m placement would be open to tuning. If, after applying the algorithm, we find that in certain regions of x the found solution vastly differs from the reference function, we could easily relocate the nodes closer to those regions (or even add extra nodes).

5.1.2. k-Plane algorithm k_m placement

For the placement of the $\{k_m\}_{m=1}^{2M}$ in the k-plane algorithm we have a lot more options. The only requirement we have is that the first M nodes, for (**??**), are placed in E^- and that the other M nodes, for (4.10), are placed in E^+.

For simplicity we limited the placement to the contours C^- and C^+ (which is permitted as they are fully within E^- and E^+, respectively). For $k \in C^-$, all terms in (4.2a) experience exponential decay. The same is true for all terms in (4.2b) for $k \in C^+$. For this reason we have chosen to implement a bound for the absolute value of our nodes $\hat{K}_{cutoff}$, which can be seen in table 5.1. Also because of this exponential decay, and because Fourier transforms tend to be concentrated around $k = 0$, it was opted to use distributions for $\{k_m\}_1^{2M}$ which are denser near $k = 0$, and more sparse as $|k|$ grows.

To achieve this we altered the equation for Chebyshev nodes, shifting parts of the equation such that the dense parts of the distribution fall at $k = 0$. For simplicity, this distribution is made with Im$k = 0$, and it will be projected onto C^- and C^+ later. For M nodes, we find

$$k_m = \begin{cases} -\hat{K}_{cutoff} + \hat{K}_{cutoff} \sin\left(\frac{\pi}{2} \frac{m-1}{\frac{M-1}{2}}\right), & m = 1, 2, \ldots, \frac{M-1}{2}, \\ 0, & m = \frac{M+1}{2}, \\ \hat{K}_{cutoff} - \hat{K}_{cutoff} \cos\left(\frac{\pi}{2} \frac{m-\frac{M+1}{2}}{\frac{M-1}{2}}\right), & m = \frac{M+1}{2} + 1, \frac{M+1}{2} + 2, \ldots, M, \end{cases} \tag{5.3}$$

where we add the restriction that M must be odd as we want to have a node at $k = 0$. This distribution can also be seen in figure 5.2. By changing the appropriate complex arguments, this distribution is then projected onto both C^- and C^+. The two distributions created by this are then used in their respective (**??**) and (4.10) for the k-plane algorithm.

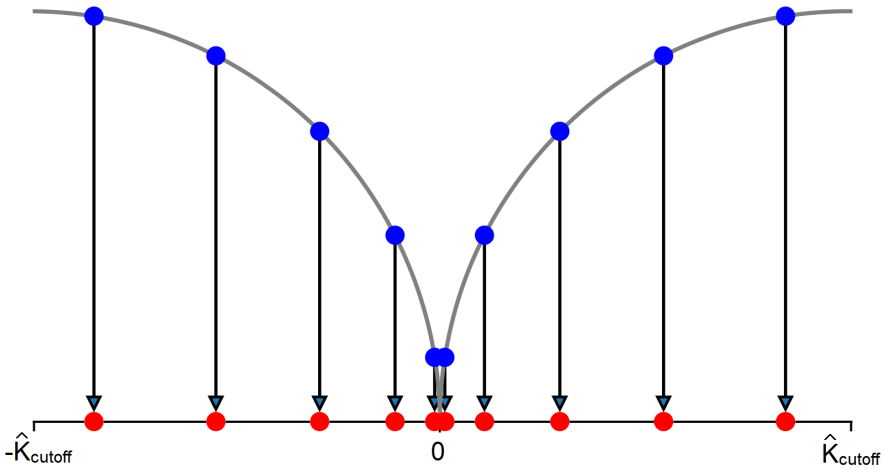


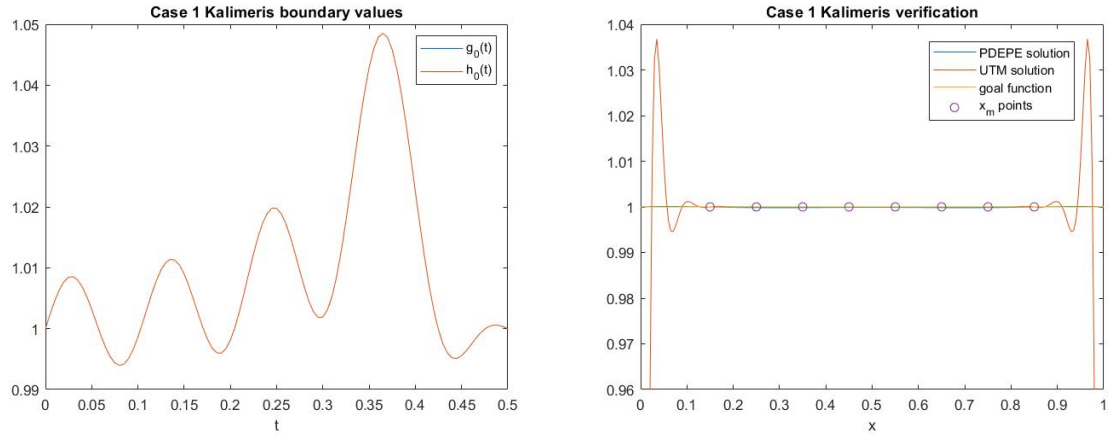
Figure 5.2: The concept of Chebyshev nodes used for a distribution more dense near $k = 0$. A more modified version of [9]

5.2. Case 1: Elevated constant value

The first scenario we will look at will be controlling $q(x, t)$ to an constant value of $q(x, T) = q_T(x) = 1$. This will be done from the initial condition $q_0(x) = -0.3 \sin(\pi x/L)$. For Case 1 the algorithms will only be used to find the Dirichlet boundary values.

We start with the Kalimeris et al. algorithm. We pick $N = 9$, $M = 8$, and, seeing as q_T has no distinguishing factors, pick the evenly spaced $\{x_m\}_1^M = 0.05 + 0.1m$. We choose the weight $w = 10^{-7}$ for our weighting matrix. We find $\beta_0 = q_T(0) = 1$ and $\alpha_0 = q_T(L) = 1$ and use these to compute our F and G matrices . Then, we solve the optimization problem (3.17). We find

$$\beta = \begin{bmatrix} 0.0147 & -0.0106 & 0.0031 & 0.0033 & -0.0067 & 0.0075 & -0.0050 & -5.5296 \times 10^{-4} & 0.0098 \end{bmatrix}^T,$$
$$\alpha = \begin{bmatrix} 0.0147 & -0.0106 & 0.0031 & 0.0033 & -0.0067 & 0.0075 & -0.0050 & -5.5296 \times 10^{-4} & 0.0098 \end{bmatrix}^T.$$



(a) TODO **(b)** TODO2

Figure 5.3: Boundary values and solutions found with the Kalimeris algorithm with uniformly distributed x_m

We notice that α and β are the same, which makes sense, as our problem is symmetric in x. We use these along with β_0 and α_0 with (3.19) to find our boundary values $g_0(t)$ and $h_0(t)$, which can be seen in figure 5.3a. These boundary values were used to calculate both $\overline{q_{T,\text{utm}}}(x)$ and $\overline{q_{T,\text{pdepe}}}(x)$. They can be seen compared to the original $q_T(x)$ in figure 5.3b.

In figure 5.3b we can see that the PDEPE solution matches q_T nicely, and that the UTM solution is indeed heavily affected by the effects of the Gibbs phenomenon. Comparing the PDEPE solution to desired q_T, we find a mean squared error of $\text{MSE}(q_T, \overline{q_{T,\text{pdepe}}}(x)) = 2.4471 \times 10^{-6}$.

We perform the Kalimeris et al. algorithm again using the Chebyshev nodes for $\{x_m\}_{m=1}^{M}$ (5.2). We keep the same N, M, and w. Again we find $\beta_0 = 1$ and $\alpha_0 = 1$ which are used to compute the F and G matrices and solving the quadratic programming, we find

$$\beta = \begin{bmatrix} 0.0132 & -0.0096 & 0.0032 & 0.0020 & -0.0045 & 0.0050 & -0.0032 & 1.6740 \times 10^{-5} & 0.0047 \end{bmatrix}^T,$$
$$\alpha = \begin{bmatrix} 0.0132 & -0.0096 & 0.0032 & 0.0020 & -0.0045 & 0.0050 & -0.0032 & 1.6740 \times 10^{-5} & 0.0047 \end{bmatrix}^T.$$

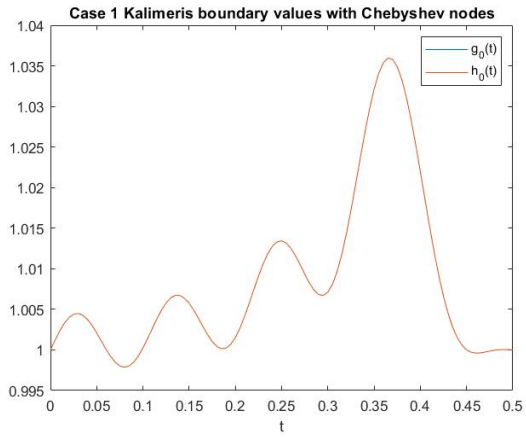
We add back the found β_0 and α_0 to compute our boundary values and the UTM and PDEPE solutions. These can be seen in figure 5.4. Comparing figures 5.3b and 5.4b we see that both solutions are close enough to q_T to see any notable difference. The means squared error of the PDEPE solution compared to the desired q_T is $\text{MSE}(q_T, \overline{q_{T,\text{pdepe}}}(x)) = 1.5637 \times 10^{-6}$ which is smaller than the MSE found with an uniform x_m distribution, but the difference is not particularly significant.

Next we apply the k-plane algorithm. This algorithm does not involve the computation of as many integrals, so we can pick higher values for N and M while keeping computation time low. We choose $N = 9$ and $M = 51$. We find $\beta_{0;0} = 1$, $\beta_{1;0} = 0$, $\alpha_{0;0} = 1$, and $\alpha_{1;0} = 0$. We use the procedure outlined in subsection 5.1.2 to find our nodes $\{k_m\}_1^{2M}$, which we use to compute $\hat{F}$ and $\hat{G}$. We split these up into their real and imaginary parts, and use these to solve the quadratic programming problem outlined in Section 4.1.2. We find

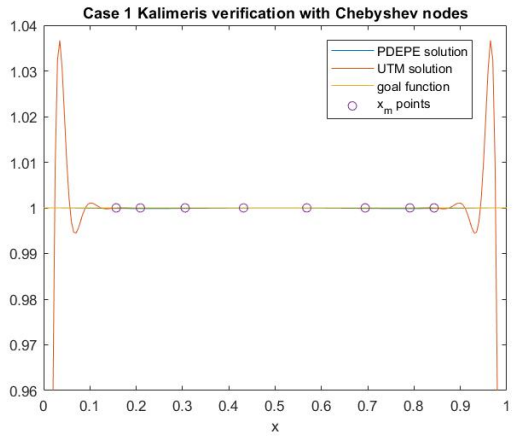
$$\beta_0 = \begin{bmatrix} -0.2546 & -0.3875 & -0.1740 & -0.1593 & -0.2330 & -0.2586 & -0.2922 & -0.1163 & 0.0094 \end{bmatrix}^T,$$
$$\beta_1 = \begin{bmatrix} -1.7242 & -0.4674 & -0.2697 & -0.4845 & -0.0450 & -0.3402 & -0.1558 & -0.1118 & -0.2284 \end{bmatrix}^T,$$
$$\alpha_0 = \begin{bmatrix} -0.2546 & -0.3875 & -0.1740 & -0.1593 & -0.2330 & -0.2586 & -0.2922 & -0.1163 & 0.0094 \end{bmatrix}^T,$$
$$\alpha_1 = \begin{bmatrix} 1.7242 & 0.4674 & 0.2697 & 0.4845 & 0.0450 & 0.3402 & 0.1558 & 0.1118 & 0.2284 \end{bmatrix}^T.$$

We use the values associated with the Dirichlet problem, β_0 and α_0, and use these to compute the UTM and the PDEPE solution. These solutions, along with the boundary values used to compute them, can be seen in figure 5.5. Comparing the PDEPE solution to the desired q_T, we find a MSE of 1.9995×10^{-6}, which is comparable to the other algorithm.

redTODO remove the xk nodes in this image All the results for case 1 are summarised in table 5.2

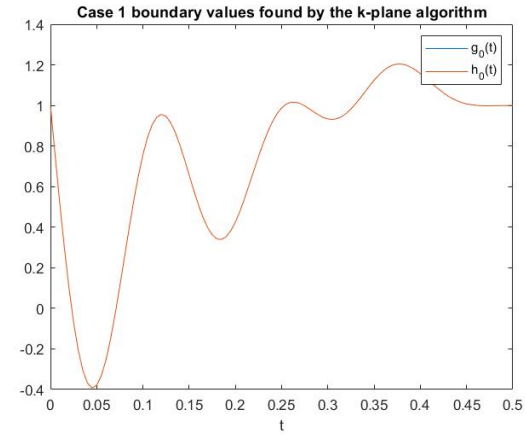


(a) boundary values found using the Kalimeris et al. algorithm

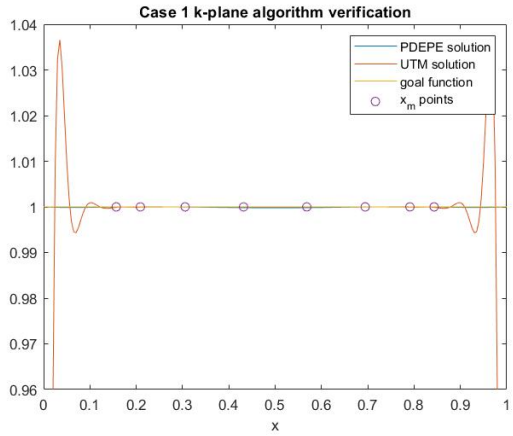


(b) Solutions to the BVP found using the Kalimeris et al. algorithm

Figure 5.4: Results found from the Kalimeris algorithm using Chebyshev nodes



(a) Boundary values found using the k-plane algorithm.



(b) Solutions found using the k-plane algorithm.

Figure 5.5: Results found using the k-plane algorithm.

	Kalimeris uniform x_m	Kalimeris Chebyshev x_m	k-plane algorithm
MSE	2.4471×10^{-6}	1.5637×10^{-6}	1.9995×10^{-6}
Computation time	$3.30s$	$3.38s$	$1.60s$

Table 5.2: Summary of the results for case 1. For the Kalemeris et al. algorithm $N = 9$, $M = 8$ is used. For the k-plane algorithm we used $N = 9$, $M = 51$

5.2.1. Smaller x_{edge} and solving without weights

Now that the first tests are performed and it is shown that the algorithms work, we feel its needed to show why some of the improvements made in Chapter 3 were needed. To do this, we will perform the Kalimeris et al. algorithm two more times, both times removing an improvement that we made.

First we will run the algorithm again with $N = 9$ and $M = 8$ (the same values as used earlier) with the uniform distribution for $\{x_m\}_{m=1}^{M}$, but we will lower the value of x_{edge} to 0.05. The solutions found with the obtained boundary values from the algorithm can be seen in figure 5.6.

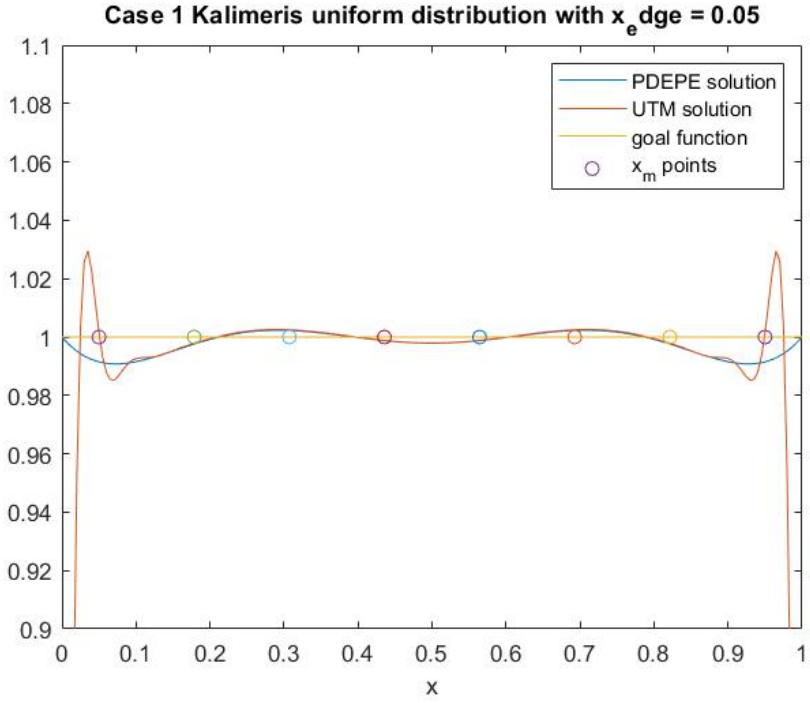
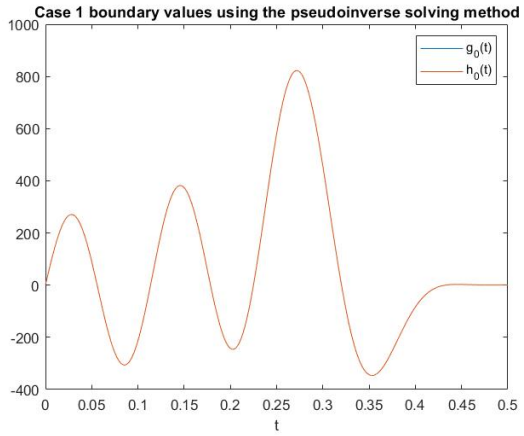


Figure 5.6: Case 1 using the Kalimeris et al. algorithm with the uniform distribution with $x_{\text{edge}} = 0.05$

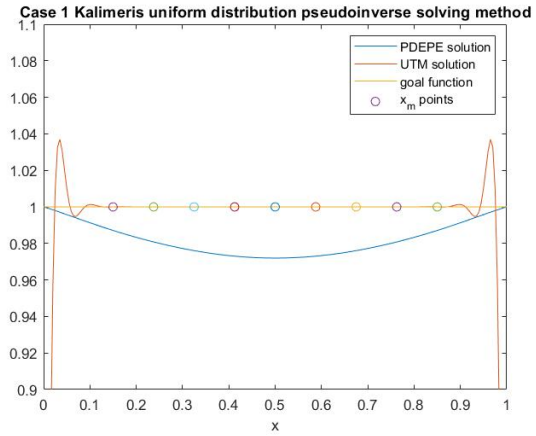
We can see in the image that the results are worse than in the case with the bigger x_{edge}. The UTM solution is matched quite well ate both the first and the last node, but the algorithm computes the incorrect values in the matrix F at these nodes, which distorts the whole solution.

The second change we undo is the solving method for γ. For this we will again use the same N, M and the uniform distribution, and we will reinstate $x_{\text{edge}} = 0.15$. We now run the algorithm again, but we find γ by using the peudoinverse of F: $\gamma = (F^TF)^{-1}F^TG$ (in [10] the regular inverse was used, as they chose $N = M$ for a square F). Note that by solving this way there are no weights present. The found boundary values and the solutions computed with these can be seen in figure 5.7.

In 5.7 we can see that the UTM solution matches the desired function quite well. The PDEPE solution, however, is not as nice. We found that this disparity between solutions generally happens when the boundary values get very large. With no weight to keep the boundary values low, they will start to affect the PDEPE solution. Besides this, boundary values of the size seen in figure 5.7a would be unusable in any real life application.



(a) Boundary values found using the pseudoinverse



(b) Solutions found using the pseudoinverse

Figure 5.7: Results found for case 1 using the Kalimeris algorithm with the pseudoinverse.

5.3. Case 2: A curved ramp

The next case we look at is a slightly more complicated $q_T(x)$ in the form of an upwards curving ramp, $q_T(x) = -0.3 + 1.8x + x.^2$. This will be done for the initial condition $q_0(x) = 0$. For this case we will use Neumann boundary values with both algorithms. With the Kalimeris algorithm we again used $N = 9$ and $M = 8$, and we used the uniform and the Chebyshev distributions for the distribution of x_m. For the k-plane algorithm we again used $N = 9$ and $M = 51$. We used a weight $w = 10^{-7}$ for both algorithms.

The algorithms were used following the same steps as in the previous section. To find the values for α_0 and β_0 we used the approximations $\beta_0 = \frac{q_T(x_{\text{step}}) - q_T(0)}{x_{\text{step}}}$ and $\alpha_0 = \frac{q_T(L) - q_T(L - x_{\text{step}})}{x_{\text{step}}}$ (since we know $q_T(x)$, we could just have taken the derivative, but the code used was set up to use data from an unknown function too). For the Kalimeris algortihm, $\alpha_0 = 3.795$ and $\beta_0 = 1.805$ were found. For the k-plane algorithm we found $\beta_{0;0} = -0.3$, $\beta_{1;0} = 1.805$, $\alpha_{0;0} = 0.5$, and $\alpha_{1;0} = 3.795$. For verification we also used the same method as in the previous section. The MSE between desired q_T and the PDEPE solutions can be seen in table 5.3, along with the computation times. The UTM and PDEPE solutions of the boundary values found using the Kalimeris algorithm with the Chebyshev distribution (the case with the worst MSE) can be seen in figure 5.8

	Kalimeris uniform x_m	Kalimeris Chebyshev x_m	k-plane algorithm
MSE	1.1133×10^{-4}	1.1245×10^{-4}	7.3594×10^{-6}
Computation time	$3.28s$	$3.26s$	$1.55s$

Table 5.3: Summary of the results for case 2.

5.4. Case 3: Sine wave with increasing frequency

For the third case we'll look at the more interesting q_T function of $q_T(x) = 0.3 \sin(4\pi x^{1.5})$. This is the first $q_T(x)$ with distinguishable features (in the form of local minima and maxima), and we will use it to test the third x_m distribution of Section 5.1.1 for the Kalimeris algorithm. This will be done using Dirichlet boundary values and with the initial condition $q_0(x) = 0$.

We start by placing nodes on the minima and maxima of $q_T(x)$. Nodes will also be placed on the points where $q_T(x)$ crosses 0. We note that $\beta_0 = \alpha_0 = 0$, which means there will be no artifacts related to the Gibbs-phenomenon. This means the limitation of not placing nodes x_{edge} away from the edges of the domain can be lifted. We find

$$x_m = \begin{bmatrix} 0.2500 & 0.3969 & 0.5200 & 0.6300 & 0.7310 & 0.8255 & 0.9148 \end{bmatrix},$$

from which we see $M = 7$ (even though we removed the x_{edge} limitation, we do not place nodes on $x = 0$ or $x = L$ as the only influence on those points is from β_0 and α_0, which we already know). We

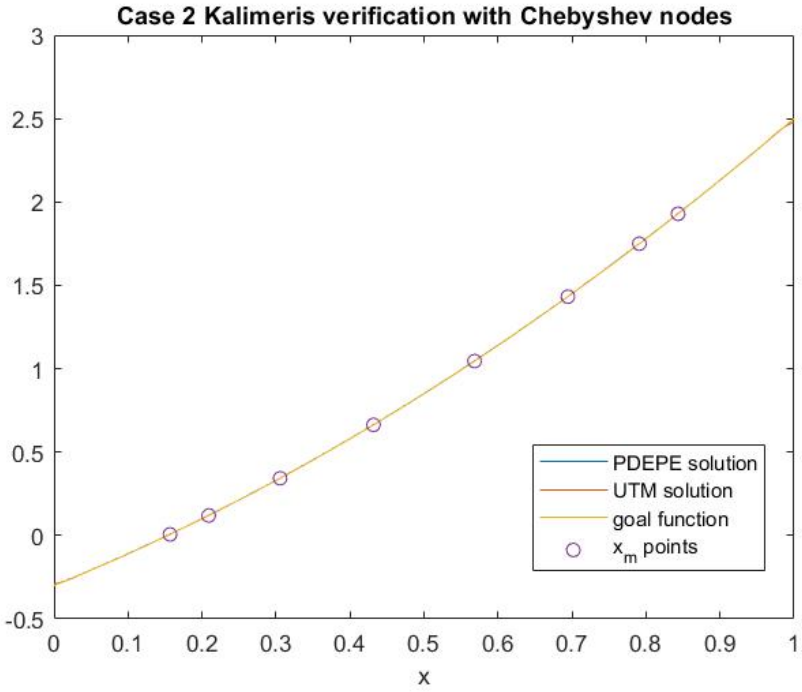


Figure 5.8: Solutions found using the Kalimeris et al. algorithm for case 2 with the Chebyshev distribution for x_m

again use $N = 9$, and use a lower weight of $w = 10^{-8}$, as the control input required to make a sine wave is quite large. The boundary values obtained by the algorithm were were used to compute the UTM and the PDEPE solution, which can be seen compared to the desired q_T in figure 5.9.

In figure 5.9 we see that while the PDEPE and UTM solutions match q_T quite well at the nodes, the replication of the full function $q_T(x)$ isn't very successful, with a MSE of $MSE = 59.251$ between the PDEPE solution $\overline{q_{T,\text{pdepe}}}(x)$ and the desired $q_T(x)$. The main disparity between the functions is located on the left half of the functions, where the distribution of x_m is the most sparse. In an attempt to improve our results, we add another node at $x = 0.1$, near the maximum of the disparity:

$$x_m = \begin{bmatrix} 0.1250 & 0.2500 & 0.3969 & 0.5200 & 0.6300 & 0.7310 & 0.8255 & 0.9148 \end{bmatrix},$$

which makes $M = 8$. We run the algorithm again to find the new boundary values. The solutions found by these can be seen in figure 5.10.

The results found with the new distribution are better, but still not great. To start off we would like to point out that the magnitude of the boundary values found for this problem is several times greater than those found in previous cases. This can be seen in the values for β and α,

$$\beta = \begin{bmatrix} -78.0529 & -26.9641 & 102.8977 & -100.4811 & 24.4558 & 69.6216 & -130.4581 & 84.0064 & 131.4533 \end{bmatrix}^T,$$
$$\alpha = \begin{bmatrix} -124.8690 & 37.6073 & 56.9298 & -98.6030 & 67.7567 & 2.4081 & -75.7427 & 81.4727 & 48.9420 \end{bmatrix}^T.$$

These are really high values when compared to desired $q_T(x)$, which has an amplitude of 0.3. With these high values we can start seeing the effects of weight w, as our solver is starting to prioritize lowering control input over reaching the desired state. This is the case in figure 5.10, where we can see that, even at the locations of the nodes, the values of the goal function and the solutions of the UTM are not the same.

We also see that the UTM solution, $\overline{q_{T,\text{utm}}}(x)$, and the PDEPE solution, $\overline{q_{T,\text{pdepe}}}(x)$, are less close than they were in the previous results. It is currently not known what causes this difference between the two solutions, but in testing it was found that this tends to happen as the magnitudes of the boundary values increase.

The uniform and the Chebyshev distributions were also tested. For both of these distributions a $x_{\text{edge}} = 0.1$ was used. The other parameters $N = 9$, $M = 8$, and $w = 10^{-8}$ were kept the same.

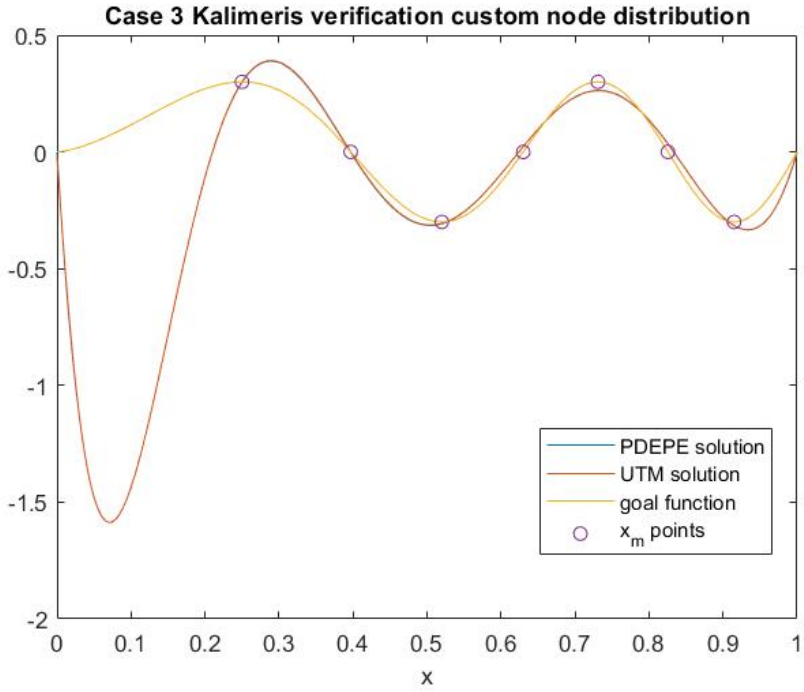


Figure 5.9: Solutions found with the Kalimeris et al. algorithm for the first iteration of the custom node distribution.

Solutions found from the obtained boundary values can be seen in figure 5.11. The MSE of these solutions can be found in table 5.4. The performance using these distributions is slightly worse, especially around $x = 0.95$.

Finally we will also apply the k-plane algorithm. Seeing the difficulties the Kalimeris algorithm had in finding the correct boundary values, we decide to immediately choose higher values for N and M. We choose $N = 40$ and $M = 71$. To accommodate the higher frequency sine wave that come with $N = 40$, we decided to lower the time step to $t_{\text{step}} = 0.0005s$. (this is mostly for the visual representation of the boundary values. Lowering the time step did very little for the found MSE.) The same weight of $w = 10^{-8}$ will be used. The solutions obtained from the found boundary values can be seen in figure REF. The MSE between the desired function and the PDEPE solution can be found in table 5.4.

Interestingly, both the MSE and the combined L_2 norm found by this application of the k-plane algorithm are lower than those found using the Kalimeris algorithm. It is highly likely this is because of the (much) higher N used in the algorithms, instead of it being because of some . TODO..

Looking at figure REF, where the boundary values obtained from the k-plane algorithm are plotted, we can see that they are indeed heavily influenced by the higher frequency sine waves.

	Kalimeris custom	Kalimeris uniform x_m	Kalimeris Chebyshev x_m	k-plane algorithm
MSE	0.5786	0.6060	0.7841	0.2679
Computation time	3.29s	3.35s	3.09s	1.97s

Table 5.4: Summary of the results for case 3.

5.4.1. Increasing N and M

In this subsection we we will investigate the effects when we increase parameters N and M, mostly done for the Kalimeris algorithm . This will be done using the same desired function $q_T(x)$ and using the uniform distribution for x_m. The results will again be measured using the MSE between the desired function q_T and the PDEPE solution computed with the found boundary values. The combined L_2 norm of both inputs, $g_0(t)$ and $h_0(t)$, were also noted. These MSE and L2 values can be seen for a varying N and M in tables 5.5 and 5.6, respectively. A weight of $w = 10^{-7}$ will be used. This higher weight

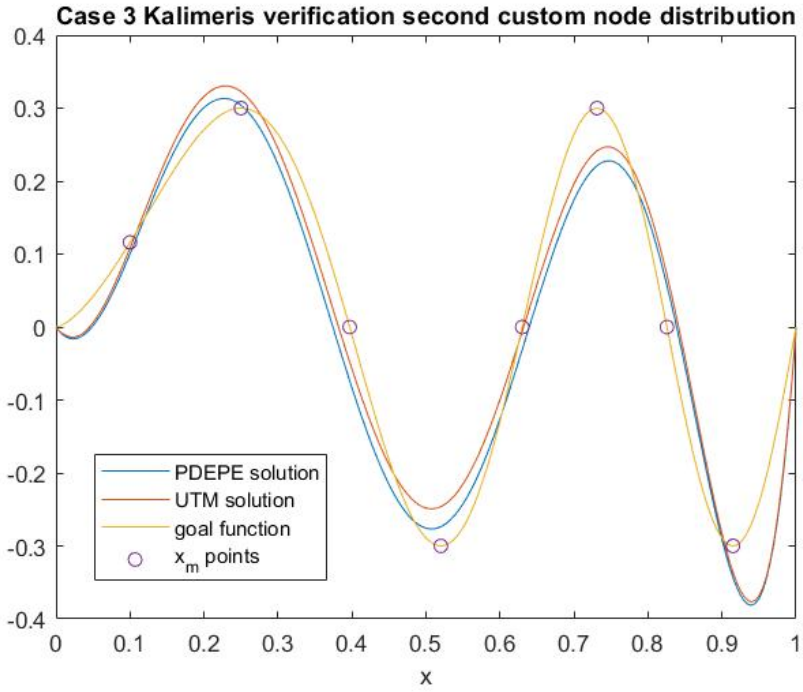


Figure 5.10: Solutions found for case 3 using the Kalimeris algorithm with the altered custom node distribution

keeps the boundary values lower which we hope should reduce the discrepancy between the UTM and PDEPE solutions, ensuring consistency between the different MSE values. Since $\alpha_0 = \beta_0 = 0$ we do not use any requirement for x_{edge}, our uniform distribution will therefore be placed between 0 and L (excluding the points 0 and L): $x_m = (m+1)L/(M+2)$ for $m = 1, 2, .., M$.

$N\backslash M$	8	9	10	12	14	16
5	0.7586	0.7292	0.7591	0.7218	0.7153	0.7321
7	0.6908	0.6356	0.6129	0.5967	0.5856	0.5809
9	0.7120	0.6265	0.5897	0.5632	0.5541	0.5530
12	0.7382	0.5788	0.5077	0.5338	0.5761	0.6494
15	0.5047	0.3487	0.3467	0.3732	0.3322	0.1592
20	0.2864	0.1853	0.1327	0.0901	0.0684	0.0595

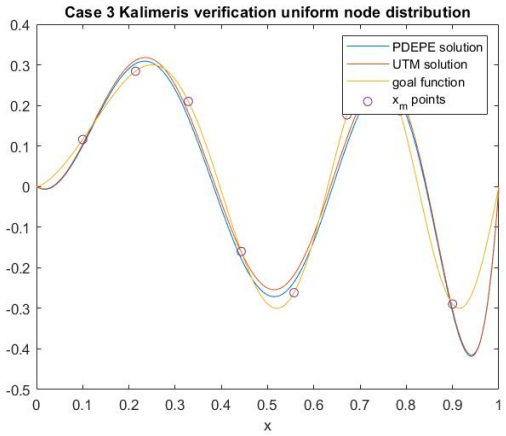
Table 5.5: MSE between the PDEPE solution and desired function for different N, M.

$N\backslash M$	8	9	10	12	14	16
5	19.4299	18.9661	18.6859	18.4183	18.3360	18.3336
7	9.8954	10.1105	10.2644	10.5831	10.9452	11.3239
9	8.0617	7.5491	7.3090	7.2854	7.5559	7.9452
12	5.1962	7.0312	8.7581	11.4767	13.4788	15.0472
15	7.1637	8.8337	9.8259	10.9168	11.5190	11.9213
20	4.0252	4.6081	5.3395	6.7890	7.9322	8.8243

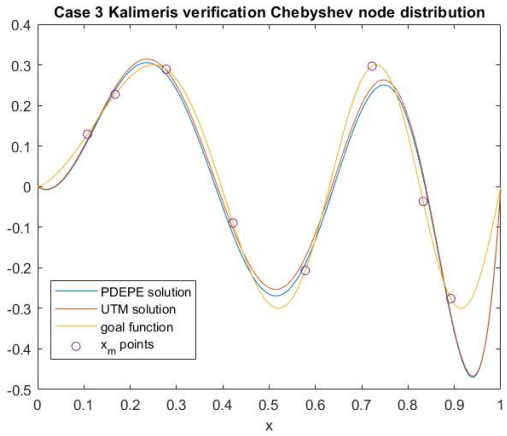
Table 5.6: L_2 norm for increasing N and M.

We can see in table 5.5 that as we increase N or M, the MSE generally goes down[1]. This is as expected, as increasing M will convey more information to the algorithm, and increasing N gives the

[1]there is an outlier for the case $N = 12$, $M = 16$. Upon inspection it was found that there is quite a separation between the UTM and PDEPE solutions here. The MSE between the UTM solution and the desired q_T was found at 0.3202.

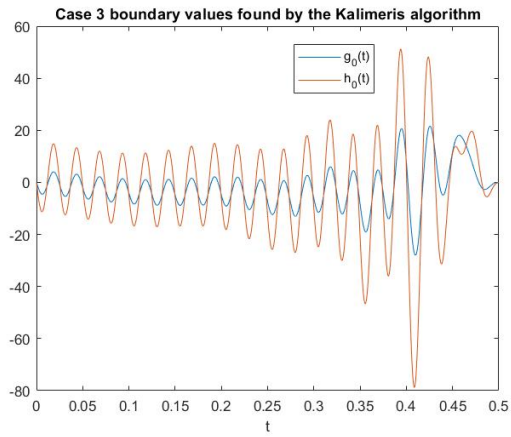


(a) Solutions found for case 3 using the Kalimeris algorithm with the uniform node distribution

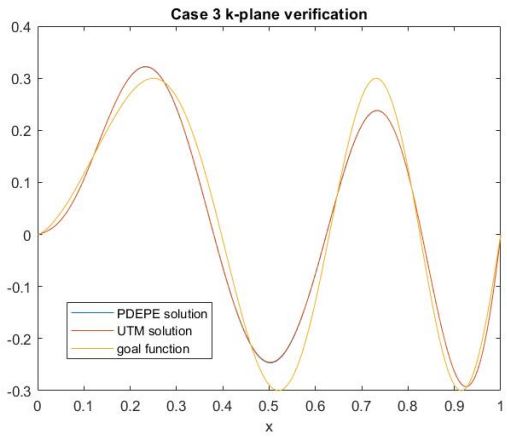


(b) Solutions found for case 3 using the Kalimeris algorithm with the Chebyshev node distribution

Figure 5.11



(a) Boundary values found using the k-plane algorithm for case 3



(b) solutions found using the k-plane algorithm for case 3

Figure 5.12

algorithm a greater range for determining the inputs. What is less expected, is the amount of reduction in L_2 norm with and increasing N. It seems that, with an increase in the amount of available basis function, the algorithm is able to find more precise boundary values with less total output. What is also interesting is that it seems that using a higher N is more important than a higher M, particularly for desired states which less natural for the heat equation to reach.

Using the Kalimeris algorithm with $N = 20$, $M = 16$, a MSE of 0.0595 was obtained. The time needed to compute the boundary values for this was 4.718 seconds. We quickly try to see if we can find the same accuracy using the k-plane algorithm. We choose $t_{\text{step}} = 0.0005s$ (this is needed for a really high N), a low weight of $w = 10^{-11}$, as it was found that the control output using the k-plane algorithm was not a problem, and $N = 100$ and $K = 271$. The boundary values for these parameters were found in 1.931 seconds. After verification we found a MSE between $\overline{q_{T,\text{pdepe}}}(x)$ and $q_T(x)$ of $MSE = 0.0501$ with a total L_2 norm of only 0.1276.

6

Conclusion

There were two main goals we tried to reach in this work. The first was to improve Kalimeris algorithm so that it is usable in more general condition. This has been achieved. The algorithm now works for arbitrary initial conditions and desired end states, for both the Neumann and Dirichlet problems. Next to that a weight has been added to limit control output. This upgraded algorithm is shown to work well for various q_T, and we see a reduction in the error as we increase the computational effect.

The second goal was to construct a competing algorithm with a similar accuracy but a lower computational cost. This has also been achieved. In all testing done the k-plane algorithm showed comparable or better results, coupled with a lower computation time.

That is not to say that this is always the better algorithm. When the wrong parameters are chosen (a too small N or weight w that is too high), the results produced by the k-plane algorithm can be quite a lot worse than any results the Kalimeris algorithm will give. This is because in the worst case, the Kalimeris algorithm will try to fit the desired solution at the nodes on the real line. We compare the Kalimeris algorithm to having training wheels on a bike: Without them you can go faster, but if you fall it will be quite a bit worse.

Regarding future work, we think there is quite a bit of optimisation to perform on the k-plane algorithm, particularly in choosing the nodes in the complex plane. While it is usable now (with some amount of verification in the process), the algorithm as given here is mainly more as a proof on concept. Because of the similarities in the solving method for γ, it might also be possible to construct a hybrid between the the algorithms, with nodes both on the complex plane and on the real line. This would be interesting to work out, as each type of node will probably have to be weighed differently.

The same algorithms can also be implemented for other PDEs. This should be relatively easy to do for linear evolution PDEs in particular, as they follow the same framework in the UTM. Besides these, we have the option to implement these techniques in Lax-integrable PDEs. These can also be solved using the UTM but currently still have the issue that the cancellation of boundary values is a nonlinear problem. The implementation of the k-plane algorithm in particular, which solves for all boundary values, could be promising.

Alternatively, there should be plenty of options for control methods using the UTM that are not founded using this type of solving algorithm. Due to the simplicity of the UTM, it should even be possible to directly compute the boundary values for any desired state q_T, although we already foresee it might be difficult to put any sort of bound of the boundary values found this way.

References

[1] Ruel V. Brown James Ward Churchill. *Complex variables and Applications*. 9. McGraw-Hill Higher Education, 2014.

[2] W.T.T Commonski. *Idealized physical setting for heat conduction in a rod with homogeneous boundary conditions.* [Online; accessed July 27, 2023], This work is licensed under the Creative Commons Attribution 3.0 International License. This can be found in http://creativecommons.org/licenses/by/3.0/. 2007. URL: https://en.wikipedia.org/wiki/Heat_equation#/media/File:Temp_Rod_homobc.svg.

[3] Bernard Deconinck, Thomas Trogdon, and Vishal Vasan. "The Method of Fokas for Solving Linear Partial Differential Equations". In: *SIAM Rev.* 56 (2014), pp. 159–186.

[4] Bernard Deconinck, Thomas Trogdon, and Xin Yang. "The numerical unified transform method for initial-boundary value problems on the half-line". In: *arXiv preprint arXiv:2006.06024* (2020).

[5] A. S Flyer N & Fokas. "A hybrid analytical–numerical method for solving evolution partial differential equations. I. The half-line". In: vol. 464. Apr. 2008, pp. 1823–1849. DOI: 10.1098/rspa.2008.0041.

[6] Athanassios S. Fokas. *A Unified Approach To Boundary Value Problems*. Society for Industrial and Applied Mathematics, 2008.

[7] The MathWorks Inc. *MATLAB Version: 9.10.0.1649659 (R2021a) Update 1*. Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com.

[8] Michael J. Hoffman Jerrold E. Marsden. *Basic Complex Analysis*. Third Edition. W. H. Freeman, 1998.

[9] S. G. Johnson. *Chebyshev Nodes*. [Online; accessed June 25, 2023], This work is licensed under the Creative Commons Attribution 4.0 International License. This can be found in http://creativecommons.org/licenses/by/4.0/. 2017. URL: https://en.wikipedia.org/wiki/Chebyshev_nodes#/media/File:Chebyshev-nodes-by-projection.svg.

[10] K. Kalimeris, T. Özsarl, and N. Dikaios. "Numerical Computation of Neumann controls for the Heat Equation on a Finite Interval". In: *IEEE Transactions on Automatic Control* (2023), pp. 1–13. DOI: 10.1109/TAC.2023.3263753.

[11] Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab. *Signals Systems (2nd Ed.)* USA: Prentice-Hall, Inc., 1996. ISBN: 0138147574.

[12] B. de Schutter. *Optimization: Linear Programming*. Delft Center for Systems and Control, TU Delft. 2020.

[13] R. J. Smith. *The Indentation Lemma*. https://tartarus.org/gareth/maths/Complex_Methods/rjs/indentation.pdf.

[14] James Stewart. *Calculus : early transcendentals*. Brooks/Cole, Cengage Learning, 2012.

[15] Cornelis Vuik et al. *Numerical Methods for Ordinary Differential Equations*. English. This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY). Netherlands: TU Delft Open, 2015. DOI: 10.5074/t.2023.001.

[16] T. van Wijk. "Literature Review". In: *[unpublished manuscript]* ().

A

Matlab code

In this appendix the code used in Matlab is given. There are two differences in notation between the code and the rest of the work namely M is K in the code (changed as k is used as the complex parameter) and the Matrix Q is G in the code.

The main file:

```
% Matlab file written for the Masters Thesis of Thijs van Wijk at the TU
% Delft DCSC.
% This file includes both the augmented Kalimeris algorithm and the k-
    Plane
% algorithm, along with both methods of verification.
clear   %tic

%Load problem specifications: (allows for easy switching between cases)
Case1         % Should have: q0_func, qT_func, Controlpoint, Controltype

% Defining constants:
stepx = 0.005;
xvals = 0:stepx:L;
tstep = 0.0005; %Make extra small, creates windowing issues otherwise
tvals = 0:tstep:T;
weight = 10^-7;
Algorithm = '1';   %Pick first or second algorithm, 1= Kalimeris, 2= k-
    plane one

% Use values for functions q. Alternatively one could use data points.
q0 = q0_func(xvals);
qT_goal = qT_func(xvals);   % plot(xvals, qT_goal)

%define algorithm parameters, phi_0 is not included in N
N=9;
K=9;    %This is M in the report. I noticed I already use k for the
    complex variable

% The following part of code dictates what kind of x_m distribution is
    used
switch controltype
    case 'Neumann'
    x_edge = 0.15;         %Don't pick anything on/too close to the edge.
        These depends on tstep and Kcutoff (also on xstep as these are
        indices for x)
```

```
30      case 'Dirichlet'    %Should be checked for afterwards
31      x_edge = 0.15;        % the Dirichlet one is bigger as effect is more
            pronounced there
32  end
33  xk = linspace(x_edge,L-x_edge,K);
34  %xk_dummy = linspace(0,L,K+2); xk = xk_dummy(2:end-1);
35  for k=1:K
36      %xk(k) = 0.5*(L) + 0.5*(L-2*x_edge)*cos(pi*(2*k-1)/(2*K)); %formula
            for Chebyshev nodes on interval [N-edge, length(xvals)-n_edge]
37
38      %xk(k) = (1/8*k)^(2/3);
39  end
40  %xk = [0.1, xk]; K=K+1;
41
42  %xk_indices = [1+n_edge, sort([index,index2]), length(xvals)-n_edge];
43  %xk = xvals(indices);
44  %K = length(xk_indices);
45
46  % More defining, includes complex contours:
47  kstep = 0.01;
48  Kcutoff = 100; %cutoff point in complex plane %Upping this gives very
        little difference
49  c = 31; %number of points used to calculate circle around pole, should be
        >~20
50  kvals = -Kcutoff:kstep:Kcutoff;
51  Cplus = [(Kcutoff:-kstep:2*kstep).*exp(1i*pi*7/8),kstep*exp(i*linspace(7*
        pi/8,pi/8,c)),(2*kstep:kstep:Kcutoff).*exp(1i*pi*1/8)]; %Note that we'
        re going around 0 here
52  Cmin = [(Kcutoff:-kstep:2*kstep).*exp(-1i*pi*1/8),kstep*exp(i*linspace(-1*
        pi/8,-pi*7/8,c)),(2*kstep:kstep:Kcutoff).*exp(-1i*pi*7/8)];%Note that
        we're going around 0 here
53  %% Computing F and G
54
55  %precomputing a_0 for g1 and h1 using the derivative, also taking a_0 for
        g0,h0
56  a0_g1 = (qT_func(stepx)-qT_func(0))/stepx;   a0_h1 = (qT_func(L)-qT_func(L
        -stepx))/stepx;
57  a0_g0 = qT_func(0);  a0_h0 = qT_func(L);
58
59  % Calculating the Fourier transform of the initial data:
60  for j=1:length(kvals)
61      hat_q0(j) = trapz(xvals,exp(-1i*kvals(j).*xvals).*q0);
62  end
63  %plot(real(dD_plus),real(hat_q0))
64
65  % Calculating \hat{q}_0(+-k) on C+ and C-
66  for j=1:length(Cplus)
67      hat_q0_Cp(j) = trapz(xvals,exp(-1i*Cplus(j).*xvals).*q0);
68      hat_q0_Cm(j) = trapz(xvals,exp(-1i*Cmin(j).*xvals).*q0);
69  end
70  Bn0 = (1-exp(-Cplus.^2*T))./(Cplus.^2); %Precalculated integral of
        tilde_phi_0*e^(-k^2T)
71
72  %Calculating parts of F
73  switch Algorithm
74  case '1'
```

```
for n = 1:N
    Bn(n,:) =  ((-1)^n*1-exp((-T)*Cplus.^2)) .* pi*n*(-T)./(Cplus.^4  .*(-T)^2 + pi^2*n^2.*ones(1,length(Cplus)));
    for k = 1:length(xk)
        switch controltype
            case 'Neumann'
            switch controlpoint
                case 'left'
                    F1(k,n) = real(trapz(Cplus,cos(Cplus.*(L-xk(k))).*1i.*Bn(n,:)./(sin(Cplus.*L))))/pi;
                case 'right'
                    F2(k,n) = real(-trapz(Cplus,cos(Cplus.*xk(k)).*1i.*Bn(n,:)./(sin(Cplus.*L))))/pi;
                case 'both'
                    F1(k,n) = real(trapz(Cplus,cos(Cplus.*(L-xk(k))).*1i.*Bn(n,:)./(sin(Cplus.*L))))/pi;
                    F2(k,n) = real(-trapz(Cplus,cos(Cplus.*xk(k)).*1i.*Bn(n,:)./(sin(Cplus.*L))))/pi;
            end
            case 'Dirichlet'
            switch controlpoint
                case 'left'
                    F1(k,n) = real(trapz(Cplus,sin(Cplus.*(L-xk(k))).*1i.*Cplus.*Bn(n,:)./(sin(Cplus.*L))))/pi;
                case 'right'
                    F2(k,n) = real(trapz(Cplus,sin(Cplus.*xk(k)).*1i.*Cplus.*Bn(n,:)./(sin(Cplus.*L))))/pi;
                case 'both'
                    F1(k,n) = real(trapz(Cplus,sin(Cplus.*(L-xk(k))).*1i.*Cplus.*Bn(n,:)./(sin(Cplus.*L))))/pi;
                    F2(k,n) = real(trapz(Cplus,sin(Cplus.*xk(k)).*1i.*Cplus.*Bn(n,:)./(sin(Cplus.*L))))/pi;
            end
        end
    end
end

switch controlpoint %take appropriate matrix
    case 'left'
        F = F1;
    case 'right'
        F = F2;
    case 'both'
        F = [F1,F2];
end

%G calculation
for k=1:length(xk) %In calculating G (Q in the text) we're calculating each integral individually as it's easier to validate this way
    Int_real(k,1) = trapz(kvals,exp(1i*kvals.*xk(k)-kvals.^2*T).*hat_q0)/2/pi;

    switch controltype
        case 'Neumann'
            Int_plus(k,1) = -trapz(Cplus,exp(1i*Cplus.*xk(k)-Cplus.^2*T)./(exp(i.*Cplus.*L)-exp(-i.*Cplus.*L)).*   (exp(1i.*Cplus.*
```

```
                L).*hat_q0_Cp + exp(-i.*Cplus.*L).*hat_q0_Cm))/2/pi;
119             Int_min(k,1) =  -trapz(Cmin ,exp(1i*Cmin.*xk(k)-Cmin.^2*T)
                    ./(exp(i.*Cmin.*L) -exp(-i.*Cmin.*L)).*    (exp(-i.*Cmin.*L
                    ).*hat_q0_Cm + exp(-i.*Cmin.*L).*hat_q0_Cp ))/2/pi;
120             Int_phi0g(k,1) = -a0_g1*real(trapz(Cplus,cos(Cplus.*(L-xk(k)))
                    .*1i.*Bn0./(sin(Cplus.*L))))/pi;
121             Int_phi0h(k,1) = a0_h1*real(trapz(Cplus,cos(Cplus.*xk(k)).*1i
                    .*Bn0./(sin(Cplus.*L))))/pi;
122         case 'Dirichlet'
123             Int_plus(k,1) = -trapz(Cplus,exp(1i*Cplus.*xk(k)-Cplus.^2*T)
                    ./(exp(-i.*Cplus.*L)-exp(i.*Cplus.*L)).*   (-exp(i.*Cplus.*
                    L).*hat_q0_Cp + exp(-i.*Cplus.*L).*hat_q0_Cm))/2/pi;
124             Int_min(k,1) = -trapz(Cmin  ,exp(1i*Cmin.*xk(k)-Cmin.^2*T)
                    ./(exp(-i.*Cmin.*L) -exp(i.*Cmin.*L)).*    (-exp(-i.*Cmin.*
                    L).*hat_q0_Cm + exp(-i.*Cmin.*L).*hat_q0_Cp ))/2/pi;
125             Int_phi0g(k,1) = -a0_g0*trapz(Cplus,sin(Cplus.*(L-xk(k))).*1i
                    .*Cplus.*Bn0./(sin(Cplus.*L)))/pi;
126             Int_phi0h(k,1) = -a0_h0*trapz(Cplus,sin(Cplus.*xk(k)).*1i.*
                    Cplus.*Bn0./(sin(Cplus.*L)))/pi;
127     end
128     qT_term(k,1) = qT_func(xk(k));
129 end
130
131 G = real(Int_real + Int_plus + Int_min - qT_term + Int_phi0g + Int_phi0h )
        ;
132 end
133 %% Algorithm 2
134 switch Algorithm
135     case '2'
136
137 %Setting k-plane parameters:
138 N_hat = 10;
139 K_hat = 71;
140 Kcutoff2 = 40; %in absolute terms
141
142 %Precomputed integrals for each Bn:
143 for n = 1:N_hat
144     Bn(n,:) =  ((-1)^n*1-exp((-T)*Cplus.^2)) .* pi*n*(-T)./(Cplus.^4  .*(-
            T)^2 + pi^2*n^2.*ones(1,length(Cplus)));
145 end
146
147 for k=1:K_hat
148 dummy_hat(k) =  0.5*(80)*cos(pi*(2*k-1)/(2*K_hat)); %formula for Chebyshev
        nodes on interval [N-edge, length(xvals)-n_edge]
149 end
150 Cmin_k = [( dummy_hat((K_hat+3)/2:end)+40).*exp(1i*pi*15/8),0,( dummy_hat
        (1:(K_hat-1)/2)-40).*exp(1i*pi*1/8)]; %"adding" the 2 here
151 Cplus_k = [( dummy_hat((K_hat+3)/2:end)+40).*exp(1i*pi*7/8),0, -(
        dummy_hat(1:(K_hat-1)/2)-40).*exp(1i*pi*1/8)]; %very beun code here but
        it works
152
153 for j=1:length(Cmin_k)     %calculating \hat{q}_0(+-k) on C-_k
154     hat_q0_Cm_k(j) = trapz(xvals,exp(-1i*Cmin_k(j).*xvals).*q0);
155     hat_qT_Cm_k(j) = trapz(xvals,exp(-1i*Cmin_k(j).*xvals).*qT_goal);
156
157     hat_q0_Cp_k(j) = trapz(xvals,exp(-1i*Cplus_k(j).*xvals).*q0);
```

```
158     hat_qT_Cp_k(j) = trapz(xvals,exp(-1i*Cplus_k(j).*xvals).*qT_goal);
159 end
160
161 % Precomputed integral for phi_0=1:
162 Bn0_k = (1-exp(-Cplus_k.^2*T))./(Cplus_k.^2); Bn0_k((end+1)/2)=T; %remove
    removable pole at k=0
163 for n = 1:N_hat
164     Bn_m =   -((-1)^n-exp(-T*Cmin_k.^2)).* pi*n*T./(Cmin_k.^4*(T)^2 + pi
        ^2*n^2);
165     Bn_p =   -((-1)^n-exp(-T*Cplus_k.^2)).* pi*n*T./(Cplus_k.^4*(T)^2 + pi
        ^2*n^2);
166
167     %Calculating all the parts of F:
168     F_hat_g0(:,n) = -1i.*Cmin_k.'.*Bn_m.';
169     F_hat_g1(:,n) = -Bn_m.';
170     F_hat_h0(:,n) = (1i.*Cmin_k.*exp(-1i.*Cmin_k.*L).*Bn_m).';
171     F_hat_h1(:,n) = exp(-1i.*Cmin_k.*L).'.*Bn_m.';
172
173     F_hat_p_g0(:,n) = -1i.*Cplus_k.'.*Bn_p.'.*exp(1i.*Cplus_k.*L).';
174     F_hat_p_g1(:,n) = -Bn_p.'.*exp(1i.*Cplus_k.*L).';
175     F_hat_p_h0(:,n) = (1i.*Cplus_k.*Bn_p).';
176     F_hat_p_h1(:,n) = Bn_p.';
177 end
178
179 % Combining all the parts of the matrix:
180 F_hat_m = [ F_hat_g0, F_hat_g1, F_hat_h0, F_hat_h1];
181 F_hat_p = [ F_hat_p_g0, F_hat_p_g1, F_hat_p_h0, F_hat_p_h1];
182 G_hat_m = (hat_qT_Cm_k - exp(-Cmin_k.^2*T).*hat_q0_Cm_k).'  -(-i*Cmin_k.*
    Bn0_k*a0_g0 - Bn0_k*a0_g1 + i*Cmin_k.*exp(-1i.*Cmin_k.*L).*Bn0_k*a0_h0
    + exp(-1i.*Cmin_k.*L).*Bn0_k*a0_h1).' ;
183 G_hat_p = (exp(1i.*Cplus_k.*L).*hat_qT_Cp_k - exp(-Cmin_k.^2*T+1i.*Cplus_k
    .*L).*hat_q0_Cp_k).'  -(-i*Cplus_k.*exp(1i.*Cplus_k.*L).*Bn0_k*a0_g0 -
    exp(1i.*Cplus_k.*L).*Bn0_k*a0_g1 + i*Cplus_k.*Bn0_k*a0_h0 + Bn0_k*a0_h1
    ).' ;
184 %plot(real(Cmin_k),real(G_hat_m)); hold on;plot(real(Cmin_k),real(G_hat_p)
    );
185 %plot(real(Cmin_k),imag(hat_qT_Cm_k)); hold on;plot(real(Cmin_k),imag(exp
    (1i.*Cplus_k.*L).*hat_qT_Cp_k ));
186 F_hat = [F_hat_m;F_hat_p];G_hat = [G_hat_m;G_hat_p]; clear F_hat_g0
    F_hat_g1 F_hat_h0 F_hat_h1 F_hat_p_g0 F_hat_p_g1 F_hat_p_h0 F_hat_p_h1;
          %clean up workspace a bit
187 end
188
189 %% Calculating a with weights
190 %con_max = 10^2;
191 W = diag(weight*ones(2*N,1));
192
193 switch controlpoint
194     case 'both'
195         %con_A = [diag(ones(2*N,1));-1.*diag(ones(2*N,1))]; %Matrices are
            twice as long with more inputs
196         %con_b = con_max.*ones(4*N,1);
197         W = diag(weight*ones(2*N,1));
198     otherwise
199         %con_A = [diag(ones(N,1));-1.*diag(ones(N,1))]; %ensuring the a,b
            values are below a certain max
```

```
200         %con_b = con_max.*ones(2*N,1);
201         W = diag(weight*ones(N,1));
202 end
203 % con_Aeq = sum(phi_n.')*tstep;
204 % con_beq = -hat_q0_0;
205
206 %a = lsqlin(F,G,con_A,con_b); %lsqlin(F,G3,con_A,con_b,con_Aeq,con_beq);
207 switch Algorithm
208     case '1'
209 %quadprog problem:
210 H = F.'*F + W;
211 f = -G.'*F;
212 a = quadprog(H,f);  %quadprog(H,f,con_A,con_b);
213     case '2'
214 W_hat = diag(weight*ones(4*N_hat,1));
215 H_hat = [real(F_hat);imag(F_hat)].'*[real(F_hat);imag(F_hat)] + W_hat;
216 f_hat = -[real(G_hat);imag(G_hat)].'*[real(F_hat);imag(F_hat)];
217 a_hat = quadprog(H_hat,f_hat);
218 end
219 % plot(real(Cmin_k),real(F_hat(1:K_hat,:)*a_hat)); hold on; plot(real(
    Cmin_k),real(G_hat(1:K_hat)));
220 % figure
221 % plot(real(Cmin_k),real(F_hat(1+K_hat:end,:)*a_hat)); hold on; plot(real(
    Cmin_k),real(G_hat(1+K_hat:end)));
222
223 %a = pinv(F)*G;
224 toc
225 %% Verification using the UTM
226
227 %Assigning k-plane vals for boundary values using earlier found Bn
228 switch Algorithm
229     case'1'
230 switch controltype
231     %Computing the transforms of the boundary values: (not actually
232     %transforming them)
233     case 'Dirichlet'
234         switch controlpoint
235             case 'left'
236                 tilde_g0 = a.'*Bn + a0_g0*Bn0; tilde_h0 = zeros(size(
                    tilde_g0));
237             case 'right'
238                 tilde_h0 = a.'*Bn + a0_h0*Bn0; tilde_g0 = zeros(size(
                    tilde_h0));
239             case 'both'
240                 tilde_g0 = a(1:N).'*Bn + a0_g0*Bn0; tilde_h0 = a(N+1:end)
                    .'*Bn + a0_h0*Bn0;
241         end
242         case 'Neumann'
243         switch controlpoint
244             case 'left'
245                 tilde_g1 = a.'*Bn + a0_g1*Bn0; tilde_h1 = zeros(size(
                    tilde_g1));
246             case 'right'
247                 tilde_h1 = a.'*Bn + a0_h1*Bn0; tilde_g1 = zeros(size(
                    tilde_h1));
248             case 'both'
```

```
249                 tilde_g1 = a(1:N).'*Bn + a0_g1*Bn0; tilde_h1 = a(N+1:end)
                        .'*Bn + a0_h1*Bn0;
250         end
251 end
252     case '2'
253         tilde_g0 = a_hat(1:N_hat).'*Bn + a0_g0*Bn0; tilde_h0 = a_hat(2*
               N_hat+1:3*N_hat).'*Bn + a0_h0*Bn0; %calcs all vals, can't do 1-
               sided control yet
254         tilde_g1 = a_hat(N_hat+1:2*N_hat).'*Bn + a0_g1*Bn0; tilde_h1 =
               a_hat(3*N_hat+1:end).'*Bn + a0_h1*Bn0;
255 end
256
257 % Calculating each integral individually again:
258 % (We are using the hat_q0's found earlier)
259 for n=1:length(xvals)
260     Int_R(n) = trapz(kvals,exp(1i*kvals.*xvals(n)-kvals.^2*T).*hat_q0)/2/
           pi;
261     switch controltype
262         case 'Neumann'
263         Int_dDp(n)      = -trapz(Cplus ,exp(1i*Cplus.*xvals(n)-Cplus.^2*T)
               ./(exp(i.*Cplus.*L)-exp(-i.*Cplus.*L)) .*(exp(i.*Cplus.*L).*
               hat_q0_Cp + exp(-i.*Cplus.*L).*hat_q0_Cm))/2/pi;
264         Int_dDm(n)      = -trapz(Cmin ,exp(1i*Cmin.*xvals(n)-Cmin.^2*T)./(
               exp(i.*Cmin.*L)-exp(-i.*Cmin.*L))  .*exp(-i.*Cmin.*L).*(
               hat_q0_Cm + hat_q0_Cp))/2/pi;
265
266         Int_dDp_copy(n) = -trapz(Cplus,exp(1i*Cplus.*xvals(n))./(exp(i.*
               Cplus.*L)-exp(-i.*Cplus.*L)) .*(-2.*exp(-i.*Cplus.*L).*tilde_g1
                + 2.*tilde_h1) )/2/pi;
267         Int_dDm_copy(n) = -trapz(Cmin ,exp(1i*Cmin.*xvals(n))./(exp(i.*
               Cmin.*L) -exp(-i.*Cmin.*L))  .*(-2.*exp(-i.*Cmin.*L).*tilde_g1
               + 2.*tilde_h1) )/2/pi;
268
269         case 'Dirichlet'
270         Int_dDp(n)      = -trapz(Cplus ,exp(1i*Cplus.*xvals(n)-Cplus.^2*T)
               ./(exp(-i.*Cplus.*L)-exp(i.*Cplus.*L)) .*(-exp(i.*Cplus.*L).*
               hat_q0_Cp + exp(-i.*Cplus.*L).*hat_q0_Cm))/2/pi;
271         Int_dDm(n)      = -trapz(Cmin ,exp(1i*Cmin.*xvals(n)-Cmin.^2*T)./(
               exp(-i.*Cmin.*L)-exp(i.*Cmin.*L))  .*exp(-i.*Cmin.*L).*(-
               hat_q0_Cm + hat_q0_Cp))/2/pi;
272
273         Int_dDp_copy(n) = -trapz(Cplus,exp(1i*Cplus.*xvals(n))./(exp(-i.*
               Cplus.*L)-exp(i.*Cplus.*L)) .*(2.*exp(-i.*Cplus.*L).*i.*Cplus.*
               tilde_g0 - 2.*i.*Cplus.*tilde_h0) )/2/pi;
274         Int_dDm_copy(n) = -trapz(Cmin ,exp(1i*Cmin.*xvals(n)) ./(exp(-i.*
               Cmin.*L) -exp(i.*Cmin.*L))  .*(2.*exp(-i.*Cmin.*L).*i.*Cmin.*
               tilde_g0 - 2.*i.*Cmin.*tilde_h0) )/2/pi;
275     end
276 end
277 qT = Int_R + Int_dDp + Int_dDp_copy + Int_dDm + Int_dDm_copy;
278
279 % figure
280 % plot(xvals,qT_goal); hold on; plot(xvals,qT);plot(xk,qT_func(xk),'o');
       legend('qT','qT goal','xk data points');
281 % title("control of the heat equation")
282
```

```
283 %% Verification using PDEPE
284 % Changing some variables to be compatible with the PDEPE functions (used
    in
285 %the next part). This is faster than also implementing the upcoming part
286 %for the k-plane algorithm.
287 switch Algorithm
288     case '2'
289         N=N_hat;
290 switch controltype
291     case 'Neumann'
292         switch controlpoint
293             case 'left'
294                 a = a_hat(N_hat+1:2*N);
295             case 'right'
296                 a = a_hat(3*N+1:4*N);
297             case 'both'
298                 a = [a_hat(N+1:2*N);a_hat(3*N+1:4*N)];
299         end
300     case 'Dirichlet'
301         switch controlpoint
302             case 'left'
303                 a = a_hat(1:N);
304             case 'right'
305                 a = a_hat(2*N+1:3*N);
306             case 'both'
307                 a = [a_hat(1:N);a_hat(2*N+1:3*N)];
308         end
309 end
310 end
311
312 % Calling the correct boundary value function: (we're doing this as we
    need
313 % to load extra variables into PDEPEs bc function)
314 switch controltype
315     case 'Neumann'
316         switch controlpoint
317             case 'left'
318                 bc = @(xl,ul,xr,ur,t) bcNeu_g_func(xl,ul,xr,ur,t,N,T,a,
                    a0_g1); %needed to pass correct data into the boundary
                    condition (something simular can to done to use data as
                     initial condition)
319             case 'right'
320                 bc = @(xl,ul,xr,ur,t) bcNeu_h_func(xl,ul,xr,ur,t,N,T,a,
                    a0_h1);
321             case 'both'
322                 bc = @(xl,ul,xr,ur,t) bcNeu_both_func(xl,ul,xr,ur,t,N,T,a,
                    a0_g1,a0_h1);
323         end
324     case 'Dirichlet'
325         switch controlpoint
326             case 'left'
327                 bc = @(xl,ul,xr,ur,t) bcDir_g_func(xl,ul,xr,ur,t,N,T,a,
                    a0_g0);
328             case 'right'
329                 bc = @(xl,ul,xr,ur,t) bcDir_h_func(xl,ul,xr,ur,t,N,T,a,
                    a0_h0);
```

```
            case 'both'
                bc = @(xl,ul,xr,ur,t) bcDir_both_func(xl,ul,xr,ur,t,N,T,a,
                    a0_g0,a0_h0);
        end
end

% PDEPE:
PDEPEsol = pdepe(0,@heatpde_func,q0_func, bc,xvals ,tvals);

% Plotting:
figure
plot(xvals,PDEPEsol(end,:)); hold on; plot(xvals,qT); %plot(xvals,qT_goal)
    ;
plot(xvals,ones(size(qT))); plot(xk,qT_func(xk),'o');
 xlabel('x'); legend('PDEPE solution','UTM solution', 'goal function','x_m
     points'); axis([0 L 0.9 1.1]);
title("Case 3 k-plane verification"); %legend('PDEPE solution','UTM
    solution', 'goal function');
title("Case 1 Kalimeris uniform distribution pseudoinverse solving method
    ")

%% Plotting Boundary vals in time
for n = 1:N
    phi_n(n,:) = sin(n*pi*(tvals)/(T));
end
g0_t = a(1:N).'*phi_n + a0_g0;
h0_t = a(N+1:2*N).'*phi_n + a0_h0;

%L2 norm:
L2 = (sqrt(sum(g0_t.^2))+sqrt(sum(h0_t.^2)))*tstep

figure
plot(tvals,g0_t); hold on; plot(tvals,h0_t); xlabel("t");  legend('g_0(t)'
    , 'h_0(t)'); %title("Case 1 boundary values found by the k-plane
    algorithm");
title("Case 1 boundary values using the pseudoinverse solving method");
%% getting the values for a in Latex
sympref("FloatingPointOutput",true);
% latex(sym(a(1:N).'))              % latex(sym(xk))
% latex(sym(a(N+1:2*N).'))
%vaf= max(0, (1-norm(y-yhat)^2/norm(y)^2)*100);
MSE = immse(PDEPEsol(end,:),ones(size(qT)))*length(qT)/L %Matlabs MSE
    takes the vector length instead of the x length
%sum((PDEPEsol(end,:)-ones(size(qT))).^2)
```

With the cases:

```
% Description: Case 1: Elevated constant value
% Should have: q0_func, qT_func, Controlpoint, Controltype

L = 1;
T = 0.5;
q0_func = @(x) -0.3*sin(pi*x/L);
qT_func = @(x) 1;

%xk = linspace(0.15,0.85,8);
controlpoint = 'both';
```

```
controltype = 'Dirichlet';
```

```
% Description: Case 2: Curved Ramp
% Should have: q0_func, qT_func, Controlpoint, Controltype

L = 1;
T = 0.5;
q0_func = @(x) 0;
qT_func = @(x) -0.3 + 1.8*x + 1*x.^2;

controlpoint = 'both';
controltype = 'Neumann';
```

```
% Description: Case 3: Sine wave
% Should have: q0_func, qT_func, Controlpoint, Controltype

L = 1;
T = 0.5;
q0_func = @(x) 0;
qT_func = @(x) 0.3*sin(pi*4*x.^1.5);
%xk = linspace(1/8,7/8,7);

controlpoint = 'both';
controltype = 'Dirichlet';
```

And the functions:

```
function [c,f,s] = heatpde_func(xvals,tvals,PDEPEsol,DuDx) %the PDE
c = 1;
f = DuDx;
s = 0;
end

function [pl,ql,pr,qr] = bcDir_both_func(xl,ul,xr,ur,t,N, T, a,a0_g0,a0_h0
    ) %Dirichlet boundary condition
for n = 1:N     %Creates the function basis
    sinbase(n) = sin(n*pi*(t)/(T));
end
g0_tval = a(1:N)'*sinbase' + a0_g0*1;   %and uses it on the found values
    for a
h0_tval = a(N+1:end)'*sinbase' + a0_h0*1;

pl = ul-g0_tval;        %And puts that into the PDE
ql = 0;

pr = ur-h0_tval;
qr = 0;
end
%Dirichlet boundary conditions for PDEPE solver. With g0(t) = g0_t, and h0
    (t) = h0_t data.
%Needs bc = @(xl,ul,xr,ur,t)  bcDir_both_func(xl,ul,xr,ur,t,N, T, a,a0_g0,
    a0_h0); defined for the
%program to add the extra variable as a boundary condition (the h1_t data)


function [pl,ql,pr,qr] = bcNeu_both_func(xl,ul,xr,ur,t,N, T, a,a0_g1,a0_h1
    ) %Neumann boundary condition
```

```
for n = 1:N      %Creates the function basis
    sinbase(n) = sin(n*pi*(t)/(T));
end
g1_tval = a(1:N)'*sinbase' + a0_g1*1;   %combines it with the found a
h1_tval = a(N+1:end)'*sinbase' + a0_h1*1;

pl = -g1_tval;
ql = 1;

pr = -h1_tval;              %and uses that data for the boundary value
qr = 1;
end
%Neumann boundary conditions for PDEPE solver. With g1(t) = 0,
%and h1(t) = h1_t data.
%Needs bc = @(xl,ul,xr,ur,t)  bcNeu_both_func(xl,ul,xr,ur,t,N, T, a,a0_g1,
   a0_h1); defined for the
%program to add the extra variable as a boundary condition (the h1_t data)
```