Lund University

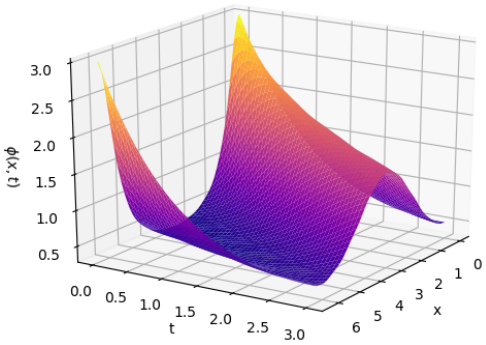
NUMK01

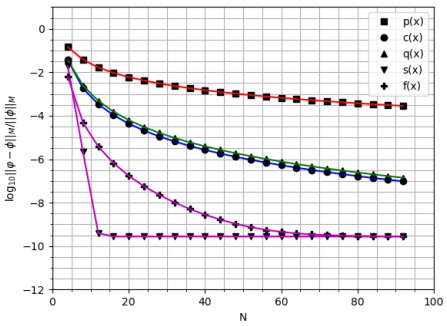
Bachelor's project in Numerical Analysis

On the Fourier Collocation Method

Author:
Henrik Lindell

Supervisor:
Philipp Birken





Abstract

This BSc thesis focuses on trying to find approximate solutions to partial differential equations using the Fourier collocation method. This method uses Fourier basis functions to approximate the solution to a partial differential equation with periodic boundary conditions. Using Fourier basis functions, one does not need to use large matrices, which makes all computations relatively fast. Another benefit is that for smooth enough initial functions, the error converges very fast. We review some theory of Fourier basis functions, some theory of circulant matrices, and the theory underlying the implementation of the Fourier collocation method. We also describe how one can improve the error of the solution by making some extra calculations before applying the Fourier collocation method, and describe three methods of time stepping. In this BSc thesis, the advection-diffusion equation is used for testing the method, as it can be explicitly solved. We finally present some numerical results. These results confirm in large parts what the theory predicts. However, for some initial functions, the error does not converge as one would expect.

Populärvetenskaplig sammanfattning

Partiella differentialekvationer är ekvationer som beskriver hur tillstånd förändrar sig. Dessa uppstår naturligt när man modellerar omvärlden, från hur molekyler sprider sig i celler till hur galaxer roterar. Att lösa dessa ekvationer är ofta näst intill omöjligt, och man behöver därför approximera lösningar med hjälp av datorer. Det här arbetet fokuserar på en speciell metod för att approximera lösningar, Fourier kollokationsmetoden. Denna metoden använder sig av periodiska funktioner (funktioner som upprepar sig med jämna mellanrum). Om man använder dessa funktioner blir alla beräkningar förhållandevis snabba, vilket är önskvärt. I detta arbete beskriver vi den matematiska teorin som underligger metoden. Vi beskriver sedan metoden och hur man kan förbättra approximationerna. Till sist presenterar vi resultatet från diverse numeriska experiment. Dessa bekräftar till stor del vad den underliggande teorin förutspår, men ett var förvånansvärt.

Acknowledgment

I would like to thank my supervisor Philipp Birken for the time he spent and for helping me with this project.

List of notation

x_j	$2\pi j/N$
e_k	e^{ikx}
$\mathcal{L}_N$	$\left\{f : f = \sum_{k=-N}^{N} e_k\right\}$
(f,g)	$\int_0^{2\pi} f(x)g^\star(x)dx$
$(f,g)_N$	$\frac{2\pi}{N}\sum_{k=0}^{N-1} f(x_k)g^\star(x_k)$
$\|\|f\|\|$	(f,f)
$\|\|f\|\|_N$	$(f,f)_N$
$\delta_{i,j}$	$\begin{cases}1, & i=j\\ 0, & i\neq j\end{cases}$
$\delta_{j,k\pm pN}$	$\begin{cases}1, & \text{if } j=k+pN \text{ of } j=-pN \text{ for some } p\in\{0,1,2,...\}\\ 0, & \text{else}\end{cases}$
$\mathcal{R}[0,2\pi]$	$\left\{g : \int_0^{2\pi} g(x)dx < \infty\right\}$
$\sum_{\|k\|=n}^{M} a_k$	$\sum_{k=-M}^{-n} a_k + \sum_{k=n}^{M} a_k$
$\hat{f}_k$	$\int_0^{2\pi} f(x)e^{-ikx}dx$
$P_N f$	$\sum_{k=-N/2}^{N/2} \hat{f}_k e^{ikx}$
τ	$\sum_{\|k\|=N/2+1}^{\infty} e_k$
$f^P(x)$	$\begin{cases}f(x-2n\pi), & x\in[2n\pi,2(n+1)\pi),\ n\in\mathbb{Z}\\ f(x), & x\in[0,2\pi)\end{cases}$
$\bar{f}(t)$	$(f(x_0,t),...,f(x_{N-1},t))^T$
e^A	$\sum_{k=0}^{\infty} A^k/k!$
c_k	$\begin{cases}1, & \|k\|<N/2\\ 2, & \|k\|=N/2\end{cases}$
$\tilde{f}_k$	$\frac{1}{2\pi}(f,e_k)_N$
$h_j(x)$	$\frac{1}{N}\sum_{k=-N/2}^{N/2}\frac{1}{c_k}e^{ik(x-x_j)}$
$I_N f$	$\sum_{j=0}^{N-1} f(x_j)h_j$
$D_{i,j}$	$h_j'(x_i)$

Table of contents

1 Introduction

1.1 Summary

This thesis is based on describing and testing the Fourier collocation method. We start in chapter 2 by showing how one in theory can use Fourier series to solve PDE's. We then review some theory of Fourier series, Fourier polynomials and Fourier basis functions. Next, in chapter 3 we present some theory of circulant matrices, which will be used in the method to advance the solution forward in time. In chapter 4 we then describe Fourier interpolation, and show how the interpolant can be calculated easily using Lagrange functions. In chapter 5 we present some results about the derivative of the Lagrange functions, and how the derivative evaluated at specific points can be calculated using a matrix-vector product, and show that the matrix is circulant and skew-symmetric. Chapter 6 is the main chapter, in which we describe the Fourier collocation method in detail, along with how one implements it to solve the advection-diffusion equation. We also show how one can reduce the error by replacing the initial function with a Fourier truncation. We also show some numerical results. In chapter 7 we describe three time stepping methods. We finally in chapter 8 show some numerical results using various time stepping methods, initial functions and time step sizes.

1.2 Motivation and Limitations

This thesis concerns the approximate solution of partial differential equations (PDE's). These arise naturally in all kinds of applications, from engineering to particle physics. One basic model PDE is the (scalar) advection-diffusion equation

$$\begin{cases} \varphi_t(x,t) + \varphi_x(x,t) = & \nu\varphi_{xx}(x,t),\ x \in [0,L],\ t \geq 0 \\ \varphi(x,0) = & \varphi_0(x) \end{cases},$$

where $\nu > 0$ is called the diffusion constant, t represents time and x represents position in along a line. This equation is commonly used to test the accuracy of numerical solution methods. The reason for this is that if one chooses a suitable initial function φ_0, the solution can be found analytically. One can then find the exact error of the numerical method. If one prescribes the boundary-value conditions

$$\varphi(0,t) = \varphi(L,t),$$

then it is natural to prescribe that the solution should be periodic in the spatial dimension x with period L, i.e. $\varphi(x,t) = \varphi(x+L,t)$. Some functions that have this property are the trigonometric functions, e.g. $\sin(2\pi nx/L)$, $\cos(2\pi nx/L)$ where n is an integer. If one chooses to only work with these functions, then one can use the rich theory of Fourier series to create methods that use this fact to their advantage.

One method, the one we focus on in this thesis, is the Fourier collocation method. When one wants to solve a PDE one first has to discretize the equation in space. This leads to the approximation of the PDE by a system of ordinary differential equations (ODE's), of the form

$$\bar{u}_t = A(t)\bar{u},$$

where $A(t)$ is a square matrix and $\bar{u}(t)$ is a column vector. In order to approximate the solution of a PDE with high accuracy, most numerical methods require large matrices. If one for example wants to approximately solve a PDE in three spatial variables, with 100 interior points in each spatial dimension, the resulting matrix is $10^6 \times 10^6$.

To calculate the solution of a linear system of equations

$$Ax = b$$

where A is a $n \times n$ matrix typically requires $\sim n^3$ operations. Therefore, approximating the solution at moderate accuracy, e.g. 100 interior points in each of three spatial dimensions, becomes unfeasible. There are two common ways of lowering the computational effort. Most methods that use large matrices are constructed so that the matrices are sparse, i.e. the overwhelming amount of elements are 0, which one can use to ones advantage. The Fourier collocation method instead uses small matrices that are full, i.e. there are not many vanishing elements. Since the matrices are small, solving the system takes very little computational effort.

The drawbacks of the Fourier collocation method, however, is that its accuracy is heavily dependent on the smoothness of the initial function. The eigenvalues of the discretized system also grow quickly with increasing matrix size. Therefore, one needs to use time stepping methods with suitable regions of absolute stability, e.g. implicit methods. One nice property of the Fourier collocation method concerns the smoothness of the solution. While e.g. finite difference methods and finite element methods typically give solutions that are piecewise affine, the Fourier collocation method gives solutions which are infinitely smooth. The accuracy of the approximation is also of exponential order, i.e. if one uses grid points with distance h, the error of the approximation is $\sim e^{-c/h}$. This can be compared to most methods whose approximations are order p, i.e. the error of the approximation is $\sim h^p$ for some integer p. Since $e^{-c/h}$ grows faster than h^p for any fixed p, this is sometimes called infinite order.

Large parts of this thesis have been constructed by studying the book of Kopriva [2]. His works can be found especially in chapters 2, 4, 6, some parts of chapters 5, and the description of Williamson's Runge-Kutta method in section 7.2. Large parts of chapter 3 come from the notes by Geller et al. [4].

2 Fourier Polynomials

In this chapter we describe some theory of Fourier series. We present some results regarding the smoothness of the initial functions. We then show that truncating the Fourier series is the best approximation in a specific norm. We then describe a discrete version of an inner product that will be used in later chapters to describe the Fourier collocation method. The theory presented in this chapter is mainly based on [2].

2.1 Fourier Series Solutions of Partial Differential Equations

One useful tool when solving PDE's are Fourier series. As an example, consider the heat equation with periodic boundary conditions

$$\begin{cases} \varphi_t(x,t) = \varphi_{xx}(x,t), & 0 < x < 2\pi \\ \varphi(x,0) = f(x), & 0 \leq x \leq 2\pi \\ \varphi(0,t) = \varphi(2\pi,t), & t \geq 0 \end{cases}. \tag{1}$$

We make an ansatz by separation of variables

$$\varphi(x,t) = \sum_{k=-\infty}^{\infty} \hat{\varphi}_k(t) e^{ikx}. \tag{2}$$

If we assume φ is continuous and piecewise C^1, we can differentiate term-wise [1]

$$\frac{\partial^2}{\partial x^2}\varphi(x,t) = \sum_{k=-\infty}^{\infty} \hat{\varphi_k}(t) \frac{d^2}{dx^2} e^{ikx} = \sum_{k=-\infty}^{\infty} -k^2 \hat{\varphi}_k(t) e^{ikx},$$

$$\frac{\partial}{\partial t}\varphi(x,t) = \sum_{k=-\infty}^{\infty} \frac{d\hat{\varphi}_k(t)}{dt} e^{ikx}.$$

By setting $\frac{\partial \varphi(x,t)}{\partial t} - \frac{\partial^2 \varphi(x,t)}{\partial x^2} = 0$, we get

$$\sum_{k=-\infty}^{\infty} \left(\frac{d\hat{\varphi}_k}{dt} + k^2 \hat{\varphi}_k \right) e^{ikx} = 0.$$

The set of functions $\left\{e^{ikx}\right\}_{k=-\infty}^{\infty}$ are linearly independent (we will show this in the next chapter), which means that each summand must vanish, leading to the system of equations

$$\frac{d\hat{\varphi}_k}{dt} + k^2 \hat{\varphi}_k = 0, \quad \forall k \in \mathbb{Z}.$$

Using the initial condition $\varphi(x,0) = f(x)$ we get the system of ordinary differential equations

$$\begin{cases} \frac{d\hat{\varphi}_k}{dt} + k^2 \hat{\varphi}_k = 0, & \forall k \in \mathbb{Z} \\ \hat{\varphi}_k(0) = g_k \end{cases}, \tag{3}$$

for some $\{g_k\}_{k=-\infty}^{\infty}$. The solutions to (3) are $\hat{\varphi}_k(t) = g_k e^{-k^2 t}$. Plugging this into (2) we get the general solution

$$\varphi(x,t) = \sum_{k=-\infty}^{\infty} g_k e^{ikx - k^2 t}.$$

Using the initial condition $\varphi(x,0) = f(x)$ we get the condition

$$f(x) = \sum_{k=-\infty}^{\infty} g_k e^{ikx}.$$

We will show later that this determines $\{g_k\}_{k=-\infty}^{\infty}$ as $g_k = \hat{f}_k$, where

$$\hat{f}_k = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-ikx}dx$$

are the Fourier coefficients of f. Finally, we get the solution

$$\varphi(x,t) = \sum_{k=-\infty}^{\infty} \hat{f}_k e^{ikx-k^2t}. \tag{4}$$

2.2 Fourier Basis Functions

As we wish to implement the solution methods numerically, Fourier series are not suitable, as we require infinite amount of information to represent the series. Instead, we limit ourselves to a linear combination of a finite number of *Fourier basis functions*, which are functions of the form $e_k(x) = e^{ikx} = \cos(kx) + i\sin(kx)$, $k \in \mathbb{Z}$. We note that $e^{ikx} = \left(e^{ix}\right)^k$. For an even natural number N we define a *Fourier polynomial of degree* $\leq N/2$ as a function satisfying

$$p(x) = \sum_{k=-N/2}^{N/2} a_k e^{ikx}$$

with $a_k \in \mathbb{C}$, $k = -N/2, -N/2+1, ..., N/2-1, N/2$. We approximate the initial function $\varphi(x,0) = f(x)$ by a Fourier polynomial of degree $\leq N/2$

$$f(x) \approx S_N(x) = \sum_{n=-N/2}^{N/2} \hat{s}_n e_n = \sum_{n=-N/2}^{N/2} \hat{s}_n e^{inx}. \tag{5}$$

The different ways of approximating f by a Fourier polynomial of degree $\leq N/2$ corresponds to different choices of $\hat{s}_n$. We denote the linear space of Fourier polynomials with degree $\leq N$ by $\mathcal{L}_N$.

Definition 1. A set of functions $\{\phi_n\}_{n=0}^{N}$ are *orthogonal* on an interval $[a,b]$ if

$$(\phi_n, \phi_m) = \int_a^b \phi_n(x)\phi_m^\star(x)dx = C_n\delta_{n,m}, \tag{6}$$

for some $C_n \in \mathbb{C}$ where $\phi_m^\star$ is the complex conjugate of ϕ_m and $\delta_{n,m}$ is the Kronecker delta which is 1 if $n = m$ and 0 otherwise.

The inner product $(\cdot,\cdot)$ induces a norm the L^2 norm on $[0, 2\pi]$

$$||\phi_n||^2 = (\phi_n, \phi_n) = \int_a^b |\phi_n(x)|^2 dx = C_n.$$

When we refer to the norm of a function, it is always this norm we are referring to, unless stated otherwise. Since the Fourier basis functions $\{e_n\}_{n=-\infty}^{\infty}$ are all periodic with period 2π, we get the following theorem:

Theorem 2. *The Fourier basis functions are orthogonal on* $[0, 2\pi]$.

Proof. Straight forward calculation gives

$$(e_n, e_m) = \int_0^{2\pi} e^{inx} e^{-imx} dx = \int_0^{2\pi} e^{i(n-m)x} = 2\pi\delta_{n,m}.$$

□

Since the basis functions are orthogonal we can determine $\hat{s}_m$ by orthogonal projection onto the space spanned by e_m,

$$(S_N, e_m) = \left(\sum_{n=-N/2}^{N/2} \hat{s}_n e_n, e_m \right) = \sum_{n=-N/2}^{N/2} \hat{s}_n (e_n, e_m) = 2\pi \hat{s}_m.$$

The coefficients in (5) are then given by

$$\hat{s}_m = \frac{1}{2\pi}(S_N, e_m), \quad |m| \leq N/2.$$

We denote the space of Riemann integrable functions on $[0, 2\pi]$ by

$$\mathcal{R}[0, 2\pi] = \left\{ g : \int_0^{2\pi} g(x)dx < \infty \right\}$$

If $f \in \mathcal{R}[0, 2\pi]$ then the representation

$$f(x) = \sum_{n=-\infty}^{\infty} \hat{f}_n e^{inx} \tag{7}$$

is unique [5]. The coefficients are then given by $\hat{f}_n = \frac{1}{2\pi}(f, e_n)$, and

$$\frac{1}{2\pi} \int_0^{2\pi} |S_N(t) - f(t)|dt \to 0, \quad \text{as } N \to \infty$$

i.e., the series converges in L^1 norm. If f is continuous and piecewise C^1, we get [1]

$$f'(x) = \sum_{n=-\infty}^{\infty} in\hat{f}_n e^{inx}.$$

We will henceforth assume that f satisfies these conditions.

2.3 Fourier Truncation

In this thesis we will discuss two ways of choosing the polynomial coefficients $\hat{s}_n$, the first one being Fourier truncation. We define the truncation operator P_N by

$$(P_N f)(x) = \sum_{k=-N/2}^{N/2} \hat{f}_k e^{ikx} = f(x) - \sum_{|k|=N/2+1}^{\infty} \hat{f}_k e^{ikx} = f(x) - \tau(x).$$

The method of Fourier truncation is then defined as replacing the initial function f by $P_N f$ for some fixed N, and approximately solving the system of ODE's (in some way

which we will not specify right now). From the definition of the truncation operator we get the truncation error

$$\tau(x) = \sum_{|k|=N/2+1}^{\infty} \hat{f}_k e^{ikx}.$$

By the orthogonality of the Fourier basis functions,$P_N\tau = 0$ and $(P_N f, \tau) = (\tau, P_N f) = 0$. We note that P_N is a projector from $\mathcal{R}[0, 2\pi]$ to $\mathcal{L}_{N/2}$, since $P_N^2 f = P_N(f - \tau) = P_N f - P_N\tau = P_N f$. We also note that $(\tau, e_n) = 0$, $|n| \leq \frac{N}{2}$. The following calculations show that truncation and differentiation commute

$$(P_N f)'(x) = \frac{d}{dx} \sum_{k=-N/2}^{N/2} \hat{f}_k e^{ikx} = \sum_{k=-N/2}^{N/2} \frac{d}{dx} \hat{f}_k e^{ikx} = \sum_{k=-N/2}^{N/2} ik\hat{f}_k e^{ikx} = (P_N f')(x). \tag{8}$$

The norm of the truncation error is given by

$$||\tau||^2 = ||f - P_N f||^2 = \int_0^{2\pi} \left(\sum_{|k|=N/2+1}^{\infty} \hat{f}_k e^{ikx} \right) \left(\sum_{|k|=N/2+1}^{\infty} \hat{f}_k^\star e^{-ikx} \right) dx =$$

$$= \sum_{|k|=N/2+1}^{\infty} \sum_{|l|=N/2+1}^{\infty} \hat{f}_k \hat{f}_l^\star \int_0^{2\pi} e^{ikx} e^{-ilx} = 2\pi \sum_{|k|=N/2+1}^{\infty} |\hat{f}_k|^2,$$

where we use the notation

$$\sum_{|k|=n}^{N} a_k = \sum_{k=-N}^{-n} a_k + \sum_{k=n}^{N} a_k. \tag{9}$$

The size of the truncation error is heavily dependent on how fast the Fourier coefficient decay. This is related to the smoothness of the periodic extension f^P of the function f, which we define as

$$f^P(x) = \begin{cases} f(x - 2n\pi), & x \in [2n\pi, 2(n+1)\pi),\ n \in \mathbb{Z} \\ f(x), & x \in [0, 2\pi) \end{cases}$$

Assume that the Fourier coefficients $\hat{f}_k$ decay as $\frac{1}{k^p}$ for some $p > 1$. One can show that the following holds [2]

$$\sum_{|k|=N/2+1}^{\infty} \frac{1}{k^{2p}} < \int_{N/2}^{\infty} \frac{2dz}{z^{2p}} = \frac{2}{2p-1} \frac{1}{(N/2)^{2p-1}}.$$

We then get for the norm of the truncation error,

$$||\tau|| < \frac{C}{(N/2)^{p-1/2}} = C \left(\frac{N}{2} \right)^{1/2-p}. \tag{10}$$

This is called *polynomial order accuracy*. As examples of this, consider the function

$$p(x) = x(2\pi - x). \tag{11}$$

We calculate the Fourier coefficients of $p(x)$. Since we are only interested in how the coefficients decay with increasing $|k|$, we can assume that $k \neq 0$ and then get

$$\int_0^{2\pi} (2\pi x - x^2) e^{ikx} dx = \left[\frac{-i}{k} (2\pi x - x^2) e^{ikx} \right]_0^{2\pi} + \frac{2\pi i}{k} \int_0^{2\pi} e^{ikx} - \frac{2i}{k} \int_0^{2\pi} x e^{ikx} dx = \frac{-4\pi}{k^2},$$

so the coefficients decay as $\frac{1}{k^2}$. We note that p^P is continuous and piecewise C^1. By noting that

$$\int_0^{2\pi} x^n e^{ikx} dx = \frac{(2\pi)^n i}{k} - \frac{i}{k} \int_0^{2\pi} n x^{n-1} e^{ikx} dx, \tag{12}$$

we can find the Fourier coefficients of polynomials recursively. Some functions, however, have coefficients that converge faster than $\frac{1}{k^p}$ for any p. If the coefficients decay as e^{-ak} for some $a > 0$, since we have

$$\sum_{|k|=N/2+1}^{\infty} |e^{-ak}|^2 < 2 \int_{N/2}^{\infty} e^{-2az} dz = \frac{1}{a} e^{-2a(N/2)}$$

we then get

$$||\tau|| < C e^{-aN/2}.$$

This is called infinite order accuracy, or *spectral accuracy.* The function

$$b(x) = \frac{3}{5 - 4\cos(x)} \tag{13}$$

is infinitely smooth with Fourier coefficients $\hat{b}_k = 2^{-|k|}$ [2], and $P_N b$ is spectrally accurate. We end this section by proving a theorem.

Theorem 3. *$P_N f$ is the best approximation from f to $\mathcal{L}_{N/2}$ in the L^2 norm on $[0, 2\pi]$, i.e.*

$$||f - P_N f|| = \min_{g \in \mathcal{L}_{N/2}} ||g - f||.$$

Note that this is what it means for a projector to be orthogonal, since $||\cdot||$ is induced by an inner product.

Proof. Consider a general approximation $S_N = \sum_{k=-N/2}^{N/2} \hat{s}_n e_n$. The norm of the approximation error is given by

$$||f - S_N||^2 = || \sum_{k=-N/2}^{N/2} (\hat{f}_k - \hat{s}_k) e_k + \sum_{|n|=N/2+1}^{\infty} \hat{f}_n e_n ||^2 = 2\pi \left(\sum_{k=-N/2}^{N/2} |\hat{f}_k - \hat{s}_k|^2 + \sum_{|n|=N/2+1}^{\infty} |\hat{f}_n|^2 \right),$$

which is minimized by setting $\hat{s}_k = \hat{f}_k$ for $|k| \leq N/2$. □

2.4 Discrete Inner Product

For many functions, we cannot integrate them analytically. We therefore seek a quadrature rule

$$Q_F[f] = \sum_{j=0}^{N} f(x_j) w_j \approx \int_a^b f(x) dx$$

for some set $\{x_j\}_{j=0}^N$ known as the nodes, and some set $\{w_j\}_{j=0}^N$ called the weights. We want functions that are orthogonal with respect to the continuous inner product to remain orthogonal with respect to the quadrature rule for as many functions as possible. Therefore we seek nodes and weights such that

$$\sum_{j=0}^{N} e^{inx_j} e^{imx_j} w_j = 2\pi \delta_{n,m}$$

for as large range of n and m as possible. We make the transformation $k = n - m$, so that the problem becomes finding nodes and weights so that

$$\sum_{j=0}^{N} e^{ikx_j} w_j = 2\pi\delta_{k,0}$$

holds for as many k as possible. For convenience we now introduce the notation $\delta_{k,\pm pN}$ as

$$\delta_{k,\pm pN} = \begin{cases} 1, & \text{if } k = pN \text{ or } k = -pN \text{ for some } p \in \{0, 1, 2, ...\} \\ 0, & \text{otherwise} \end{cases}, k \in \mathbb{Z}.$$

Theorem 4. *For $N \in \mathbb{Z}_+$ and $k \in \mathbb{Z}$ we have*

$$\sum_{j=0}^{N-1} e^{ik\left(\frac{2\pi j}{N}\right)} = N\delta_{k,\pm pN}.$$

where $\delta_{k\pm pN}$ is defined as above.

Proof. It is true for k being any multiple of N, since then $e^{ik\left(\frac{2\pi j}{N}\right)} = 1$. Then the left hand side is $\sum_{j=0}^{N-1} 1 = N$, and the right hand side is N (by the previously introduced notation). If k is not a multiple of N, then by calling $\xi = e^{2\pi ik/N}$ we get (since the sum is a geometric sum)

$$\sum_{j=0}^{N-1} e^{ik\left(\frac{2\pi j}{N}\right)} = \sum_{j=0}^{N-1} \xi^j = \frac{1 - \xi^N}{1 - \xi} = \frac{1 - e^{2\pi ik}}{1 - e^{2\pi ik/N}} = 0.$$

□

The theorem above suggests that we use the nodes $x_j = \frac{2\pi j}{N}$ henceforth. x_j will refer to these nodes. Since $e^{ix_0} = e^{ix_N} = 1$, we ignore the term $j = N$, and normalize to get $w_j = \frac{2\pi}{N}$. Then, by theorem 4 we have

$$\frac{2\pi}{N} \sum_{j=0}^{N-1} e^{ikx_j} = 2\pi\delta_{k,\pm pN}.$$

In particular, this exactly matches the integral when $p = 0$

$$\frac{2\pi}{N} \sum_{j=0}^{N-1} e^{ikx_j} = 2\pi\delta_{k,0} = \int_0^{2\pi} e^{ikx} dx, \quad |k| = 0, 1, ..., N - 1.$$

We define the *Fourier quadrature rule* by

$$Q_F[f] = \frac{2\pi}{N} \sum_{j=0}^{N-1} f(x_j), \quad x_j = \frac{2\pi j}{N}.$$

This is just the composite trapezoidal rule when f is periodic. Since $k = n - m$ we get

$$\frac{2\pi}{N} \sum_{j=0}^{N-1} e^{inx_j} e^{-imx_j} = \int_0^{2\pi} e^{inx} e^{-imx} dx, \quad |n - m| = 0, 1, ..., N - 1.$$

We next define the *discrete inner product*.

Definition 5. For $N \in \mathbb{Z}_+$, we define the discrete inner product as

$$(u, v)_N = \frac{2\pi}{N} \sum_{j=0}^{N-1} u(x_j) v^\star(x_j), \; x_j = \frac{2\pi j}{N}. \tag{14}$$

Then the following holds

$$(e_n, e_m)_N = (e_n, e_m) = 2\pi\delta_{n,m}, \quad |n-m| = 0, 1, ..., N-1$$
$$||e_n||_N^2 = 2\pi.$$

The Fourier quadrature rule integrates the Fourier basis functions e_k exactly for $|k| = 0, 1, ..., N-1$. There is a difference between the continuous inner product (6) and the discrete inner product (14) however, since

$$(e_n, e_m) \neq 0 \iff n = m$$

while

$$(e_n, e_m)_N \neq 0 \iff n - m = pN \text{ or } n - m = -pN \text{ for some } p \in \{0, 1, 2, ...\}.$$

This phenomenon is connected to aliasing errors, which we will discuss in later chapters. We note that $(u, v)_N$ reminds us of the Euclidean inner product for real vectors

$$(\bar{u}, \bar{v})_2 = \bar{u}^T \bar{v} = \sum_{k=1}^{N} u_k v_k.$$

We therefore define to a function f the column vector $\bar{f}$ by

$$\bar{f}(t) = \begin{pmatrix} f(x_0, t) \\ f(x_1, t) \\ \cdot \\ \cdot \\ \cdot \\ f(x_{N-1}, t) \end{pmatrix}.$$

From this we can rewrite the discrete inner product, assuming u and v are both real-valued functions,

$$(u, v)_N = \frac{2\pi}{N} (\bar{u}, \bar{v})_2 = \frac{2\pi}{N} \bar{u}^T \bar{v}.$$

3 Circulant matrices

Many of the matrices used in the implementation of the methods here are *circulant* matrices. We show below that all circulant matrices have the same eigenvectors, and all circulant matrices can be decomposed into $U^\star \Psi U$, where U is unitary and Ψ is diagonal. We also show that products, sums and inverses of circulant matrices are circulant. The majority of the theory is based on [4].

3.1 Definition and Eigenvalues

We start by defining what a circulant matrix is and proving a theorem about the eigenvalues and eigenvectors of a circulant matrix.

Definition 6. A circulant matrix $\text{circ}\{a_0, a_2, ..., a_{n-1}\}$ is a matrix $A \in \mathbb{C}^{n\times n}$ of the form

$$A = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & ... & a_{n-2} & a_{n-1} \\ a_{n-1} & a_0 & a_1 & a_2 & ... & a_{n-3} & a_{n-2} \\ a_{n-2} & a_{n-1} & a_0 & a_1 & ... & a_{n-4} & a_{n-3} \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ . & . & . & . & & . & . \\ a_1 & a_2 & a_3 & a_4 & ... & a_{n-1} & a_0 \end{pmatrix}.$$

Theorem 7. *Let $A = circ\{a_0, ..., a_{n-1}\}$. Then A has the eigenvalues*

$$\lambda_l = \sum_{j=0}^{n-1} e^{2\pi ijl/n} a_j$$

with corresponding eigenvectors

$$\bar{u}_l = \begin{pmatrix} 1 \\ e^{2\pi il/n} \\ e^{4\pi il/n} \\ . \\ . \\ . \\ e^{2(n-1)\pi il/n} \end{pmatrix}$$

for $l = 0, 1, ..., n-1$.

Proof. Note that A is the matrix

$$\begin{pmatrix} a_0 & a_1 & ... & a_{n-1} \\ a_{n-1} & a_0 & ... & a_{n-2} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_1 & a_2 & ... & a_0 \end{pmatrix},$$

so the k'th row of A is $A_k = (a_{n-k+1}, a_{n-k+2}, ..., a_{n-1}, a_0, a_1, ..., a_{n-k})$, and the k'th element of $\bar{u}_l$ is $\bar{u}_{lk} = e^{2\pi(k-1)il/n}$. By noting that $e^{2\pi inl/n} = (e^{2\pi i})^{-l} = 1$, we calculate the k'th element of $A\bar{u}_l$ to be

$$(A\bar{u}_l)_k = a_{n-k+1} + e^{2\pi il/n} a_{n-k+2} + ... + e^{2\pi i(k-2)l/n} a_{n-1} + e^{2\pi i(k-1)l/n} a_n =$$

$$\sum_{s=n-k+1}^{n-1} e^{-2\pi i(n-k+1)l/n} e^{2\pi isl/n} a_s + \sum_{j=0}^{n-k} e^{2\pi i(k-1)l/n} e^{2\pi ijl/n} a_j =$$

$$e^{2\pi i(k-1)l/n} \left(\sum_{s=n-k+1}^{n-1} e^{-2\pi in/n} e^{2\pi isl/n} a_s + \sum_{j=0}^{n-k} e^{2\pi ijl/n} a_j \right) =$$

$$e^{2\pi i(k-1)l/n} \left(\sum_{s=n-k+1}^{n-1} e^{2\pi isl/n} a_s + \sum_{j=0}^{n-k} e^{2\pi ijl/n} a_j \right) = e^{2\pi i(k-1)l/n} \sum_{j=0}^{n-1} e^{2\pi ijl/n} a_j = \bar{u}_{l,k} \lambda_l.$$

Since this holds for $k = 1, ..., n$, we get $A\bar{u}_l = \lambda_l \bar{u}_l$. The matrix with rows $\bar{u_1}, ..., \bar{u_n}$ is a Vandermonde matrix, so the determinant is nonzero [6] and the n vectors $\bar{u_1}, ..., \bar{u}_n$ are therefore linearly independent, and form a basis for $\mathbb{C}^n$. The λ_l described above are therefore the n eigenvalues. □

3.2 Eigenvalue Decomposition

If we have two $N \times N$ circulant matrices, theorem 7 applies to each of them. Then the matrices have the same eigenvectors. Since all linearly independent eigenvectors are determined by theorem 7. This gives rise to the following theorem.

Theorem 8. *Any $N \times N$ circulant matrix C can be decomposed into $C = U^\star \Psi U$, where $U^\star_{j,k} = e^{2\pi ijk/N}/\sqrt{N}$, $j, k = 0, 1, ..., N-1$ is a unitary matrix and Ψ is diagonal.*

Proof. We first note that the eigenvectors $\bar{u}_l = (1, e^{2\pi il/n}, e^{4\pi il/n}, ..., e^{2(n-1)\pi il/n})^T$. This shows that the normalized eigenvectors $\bar{y}_l = \frac{1}{\sqrt{N}}(1, e^{2\pi il/N}, ..., e^{2\pi il(N-1)/N})^T$ are orthonormal. Then the k'th column of $U^\star$ is $\bar{y}_l$. Now, let Ψ be the diagonal matrix with element $\Psi_{l,l} = \lambda_l$. Let C be a circulant matrix. Then we have

$$CU^\star = U^\star \Psi.$$

Since the columns of $U^\star$ are orthonormal, U is unitary. Therefore

$$C = U^\star \Psi U.$$

□

The theorem above also shows that a matrix $A = U^\star \Psi U$, with U given in theorem 8 and Ψ a diagonal matrix, is circulant. To explain this we note that the formula of the eigenvalues in theorem 7 gives

$$\begin{pmatrix} \lambda_0 \\ . \\ . \\ . \\ \lambda_{N-1} \end{pmatrix} = \sqrt{N} U^\star \begin{pmatrix} a_0 \\ . \\ . \\ . \\ a_{N-1} \end{pmatrix} \iff \begin{pmatrix} a_0 \\ . \\ . \\ . \\ a_{N-1} \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} \lambda_0 \\ . \\ . \\ . \\ \lambda_{N-1} \end{pmatrix}.$$

Given a square matrix $U^\star \Psi U$, where Ψ is a diagonal matrix with entries $\Psi_{l,l} = \lambda_l$, the equation above gives us entries $a_0, ..., a_{N-1}$ such that $C = circa_0, ..., a_{N-1}$ is a circulant matrix with the eigenvalues $\lambda_0, ..., \lambda_{N-1}$. Then $C = U^\star \Psi U$. Therefore, $U \star \Psi U$ is a circulant matrix. From this we get some properties of circulant matrices in the following theorem.

Theorem 9. *Let A and B be circulant matrices. Then $A + B$, $AB = BA$ and A^{-1} are all circulant.*

Proof. The statements are easily shown by writing $A = U^\star \Psi_A U$ and $B = U^\star \Psi_B U$. We then get

$$A + B = U^\star \Psi_A U + U^\star \Psi_B U = U^\star (\Psi_A + \Psi_B) U,$$

$$AB = U^\star \Psi_A U U^\star \Psi_B U = U^\star \Psi_A \Psi_B U.$$

We note that $\Psi_A + \Psi_B$ and $\Psi_A \Psi_B$ are diagonal. Therefore, $A+B$ and AB are circulant. Since diagonal matrices commute, we get $\Psi_A \Psi_B = \Psi_B \Psi_A$, so $AB = BA$. We make the ansatz

$$A^{-1} = U^\star \Psi_A^{-1} U.$$

This is the inverse of A, since

$$A U^\star \Psi_A^{-1} U = U^\star \Psi_A U U^\star \Psi_A^{-1} U = U^\star \Psi_A \Psi_A^{-1} U = U^\star U = I.$$

□

Note that the theorem above states that the set of circulant $N \times N$ matrices is a commutative ring. The eigenvalues of a circulant matrix C can be found by

$$\Psi = U U^\star \Psi U U^\star = U C U^\star.$$

Assume that we want to solve the system of ordinary differential equations

$$\bar{u}_t = A\bar{u}$$

where A is a $N \times N$ circulant matrix. We approximate the solution on the time steps t_n with $u^n = u(t_n)$ using the explicit Euler method

$$u^{n+1} = u^n + \Delta t A u^n = (I + \Delta t A) u^n.$$

Applying this iteratively, we get

$$u^n = (I + \Delta t A)^n u^0$$

Let us now say that we want to calculate the solution for $t = N\Delta t$, giving the approximation $\bar{u}(t) \approx (I + \Delta t A)^N u^0$. Taking the limit as $N \to \infty$ while keeping t constant we get

$$\bar{u}(t) = \lim_{N \to \infty} (I + \frac{t}{N})^N u^0 = e^{tA} u^0,$$

where the matrix exponential is defined as

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k$$

Calculating the matrix exponential for a general square matrix M is hard. One needs to find matrices V and Λ such that

$$M = V \Lambda V^{-1}.$$

Then

$$e^{tM} = \sum_{k=0}^{\infty} \frac{1}{k!} t^k M^k = \sum_{k=0}^{\infty} \frac{1}{k!} t^k (V \Lambda V^{-1})^k = \sum_{k=0}^{\infty} \frac{1}{k!} t^k V \Lambda^k V^{-1} = V \left(\sum_{k=0}^{\infty} \frac{1}{k!} t^k \Lambda^k \right) V^{-1} =$$

$$Ve^{t\Lambda}V^{-1}.$$

If M is diagonalizable, one can take the columns of V to be the normalized eigenvectors of M and take Λ to be a diagonal matrix whose diagonal entries are the eigenvalues of M. If M is not diagonalizable, finding V and Λ is even more difficult, involving finding the generalized eigenvectors of M. However, since our matrix A is circulant, we know that

$$A = U^{\star}\Psi_A U,$$

with U already known and the entries of Ψ_A being easily calculated using theorem 7. We then get

$$e^{tA} = U^{\star}e^{t\Psi_A}U.$$

The fact that we can easily calculate e^{tA} for any t if A is circulant will mean that we can solve the system of ODE's arising from the spatial discretization of the PDE exactly, meaning that there will be no error due to time stepping (if we choose to use this method of solving the system of ODE's).

4 Fourier Interpolation

In this chapter we describe the interpolation of a periodic function using Fourier basis functions, which is the second way of choosing polynomial coefficients (the first way being through Fourier truncation). We show how the interpolation can be calculated using the discrete inner product, and how this gives rise to aliasing errors. We finally show that unlike Fourier truncation, interpolation and differentiation do not commute. The theory is based on [2].

4.1 Discrete Fourier Coefficients

We define the Fourier interpolant $I_N f$ of a real-valued function f as the Fourier polynomial of degree $\leq N/2$ satisfying the interpolation condition

$$I_N f(x_j) = f(x_j),\ j = 0, 1, ..., N-1.$$

[2] proposes using the *discrete Fourier coefficients*

$$\tilde{f}_k = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)e^{-ikx_j} = \frac{1}{2\pi}(f, e_k)_N, \quad |k| \leq N/2. \tag{15}$$

We first define the *Lagrange function* and then prove a lemma. Some readers might recognize this as the inverse discrete Fourier transform (DFT).

Definition 10. The Lagrange functions are the set of functions $\{h_j\}$ defined by

$$h_j(x) = \frac{1}{N}\sum_{k=-N/2}^{N/2} \frac{1}{c_k} e^{ik(x-x_j)}, \quad j = 0, 1, ..., N-1,$$

where

$$c_k = \begin{cases} 1, & |k| < N/2 \\ 2, & |k| = N/2 \end{cases}$$

Lemma 11. *For the Lagrange functions h_j evaluated at the nodes x_n, the following holds*

$$h_j(x_n) = \frac{1}{N}\sum_{k=-N/2}^{N/2} \frac{1}{c_k} e^{2\pi i(n-j)k/N} = \delta_{j,n}.$$

Proof. We note that

$$\begin{aligned} e^{2\pi i(n-j)\left(\frac{N}{2}\right)/N} = e^{i\pi(n-j)} &= (-1)^{n-j}, \\ e^{2\pi i(n-j)\left(\frac{-N}{2}\right)/N} = e^{-i\pi(n-j)} &= (-1)^{n-j}. \end{aligned}$$

We can then identify the $k = \frac{N}{2}$ with the $k = \frac{-N}{2}$ term, and since

$$\frac{1}{c_{\frac{N}{2}}} = \frac{1}{c_{\frac{-N}{2}}} = \frac{1}{2}$$

we get (by making the transformation $l = k + N/2$)

$$\frac{1}{N}\sum_{k=-N/2}^{N/2} \frac{1}{c_k} e^{2\pi i(n-j)k/N} = \frac{1}{N}\sum_{k=-N/2}^{N/2-1} e^{2\pi i(n-j)k/N} = \frac{1}{N}\sum_{l=0}^{N-1} e^{2\pi i(n-j)(l-N/2)/N} =$$

$$= \frac{1}{N}\sum_{l=0}^{N-1} e^{2\pi i(n-j)l/N} e^{-i\pi(n-j)} = \frac{(-1)^{n-j}}{N}\sum_{l=0}^{N-1} e^{2\pi i(n-j)l/N} = (-1)^{n-j}\delta_{n-j,\pm pN} =$$

$$= (-1)^{n-j}\delta_{n,j} = \delta_{n,j}$$

where the third to last equality comes from theorem 4, and the second to last equality comes from the fact that $|n-j| \leq N-1$. □

We can now prove a theorem about how to explicitly calculate $I_N f$.

Theorem 12.

$$I_N f = \sum_{j=0}^{N-1} f(x_j) h_j,$$

and $I_N f$ is the only Fourier polynomial of degree $\leq N/2$ that satisfy the interpolation condition $I_N f(x_j) = f(x_j)$, $j = 0, 1, ..., N-1$.

Proof. We note that since each h_j is a Fourier polynomial of degree $\leq N/2$, the sum stated above is also a Fourier polynomial of degree $\leq N/2$. Using lemma 11 we calculate the sum at the nodes

$$I_N f(x_n) = \sum_{j=0}^{N-1} f(x_j) h_j(x_n) = \sum_{j=0}^{N-1} f(x_j)\delta_{j,n} = f(x_n), \tag{16}$$

so the sum satisfies the interpolation condition. For uniqueness of the Fourier interpolant, we write a Fourier polynomial of degree $\leq N/2$ in the form

$$S_N = \sum_{k=-N/2}^{N/2} a_k e^{ikx}$$

with

$$S_N(x_j) = \sum_{k=-N/2}^{N/2} a_k e^{ikx_j} = f(x_j).$$

We note that since S_N satisfies the interpolation condition, then

$$\begin{pmatrix} e^{-i(N/2)x_0} & e^{-i(N/2-1)x_0} & ... & e^{i(N/2)x_0} \\ e^{-i(N/2)x_1} & e^{-i(N/2-1)x_1} & ... & e^{i(N/2)x_1} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ e^{-i(N/2)x_{N-1}} & e^{i(N/2-1)x_{N-1}} & ... & e^{i(N/2)x_{N-1}} \end{pmatrix} \begin{pmatrix} a_{-N/2} \\ a_{-N/2+1} \\ . \\ . \\ . \\ a_{N/2} \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ . \\ . \\ . \\ f(x_{N-1}) \end{pmatrix}.$$

Note that the matrix is a multiple of $U^\star$ with the rows permuted. Since $\det(U^\star) \neq 0$, the determinant of the matrix is also non-zero, so the system has a unique solution for each right-hand side. Therefore, the Fourier polynomial of degree $\leq N/2$ satisfying the interpolation condition is unique. Therefore, since the sum above satisfies the interpolation condition, it is the Fourier interpolant. □

We can also write $I_N f$ in the same form as $P_N f$ by writing

$$I_N f(x) = \sum_{j=0}^{N-1} h_j(x) f(x_j) = \sum_{j=0}^{N-1} \left(\frac{1}{N} \sum_{k=-N/2}^{N/2} \frac{1}{c_k} e^{ik(x-x_j)} \right) f(x_j) =$$

$$\sum_{k=-N/2}^{N/2}\left(\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)e^{-ikx_j}\right)\frac{e^{ikx_j}}{c_k}=\sum_{k=-N/2}^{N/2}\frac{\tilde{f}_k}{c_k}e^{ikx}$$

where

$$\tilde{f}_k=(f,e_k)_N$$

are called the *discrete Fourier coefficients*.

We note that I_N is also a projector, but not an orthogonal one. The discrete Fourier coefficients are N-periodic since

$$\tilde{f}_{k\pm N}=\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)e^{-i(k\pm N)x_j}=\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)e^{-ikx_j}e^{\mp iN2\pi j/N}=$$

$$=\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)e^{-ikx_j}e^{\mp 2\pi ij}=\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)e^{-ikx_j}=\tilde{f}_k.$$

In particular, $\tilde{f}_{-N/2}=\tilde{f}_{N/2}$. Also, when f is real-valued, $\tilde{f}_{N/2}$ is also real, since

$$\tilde{f}_{N/2}=\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)e^{-i\frac{N}{2}x_j}=\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)e^{i-\pi j}=\frac{1}{N}\sum_{j=0}^{N-1}f(x_j)(-1)^j.$$

By definition, $(I_Nf)=f$ at the nodes, leading to

$$(I_Nf,e_k)_N=(f,e_k)_N.$$

We end this section with a theorem about the discrete inner product

Theorem 13. *Suppose u and v are two Fourier interpolants, possibly of different functions, of degrees $\leq N$. Then $(u,v)_N=(u,v)$*

Proof. We write the interpolants in their modal form

$$u=\sum_{k=-N/2}^{N/2}\frac{\tilde{a}_k}{c_k}e_k,\quad v=\sum_{l=-N/2}^{N/2}\frac{\tilde{b}_l}{c_l}e_l.$$

We have that

$$u(x_j)=\sum_{k=-N/2}^{N/2}\frac{\tilde{a}_k}{c_k}e^{ikx_j}=\sum_{k=-N/2}^{N/2-1}\frac{\tilde{a}_k}{c_k}e^{ikx_j}+\frac{\tilde{a}_{N/2}}{c_{N/2}}e^{i\frac{N}{2}x_j}=$$

$$\sum_{k=-N/2}^{N/2-1}\frac{\tilde{a}_k}{c_k}e^{ikx_j}+\frac{\tilde{a}_{-N/2}}{c_{-N/2}}e^{-i\frac{N}{2}x_j}=\sum_{k=-N/2}^{N/2-1}\tilde{a}_ke^{ikx_j}.$$

where the last equality holds since $c_n=1$ for $|k|<N/2$ and $c_{-N/2}=2$. The corresponding equality holds for v. Therefore

$$(u,v)_N=\sum_{k=-N/2}^{N/2-1}\sum_{l=-N/2}^{N/2-1}\tilde{a}_k\tilde{b}_l(e_k,e_l)_N=\sum_{k=-N/2}^{N/2-1}\sum_{l=-N/2}^{N/2-1}\tilde{a}_k\tilde{b}_l(e_k,e_l)=(u,v).$$

□

4.2 Error of Fourier Interpolation

Since the Fourier truncation $P_N f$ is the best approximation from f to $\mathcal{L}_{N/2}$ in the L^2 norm on $[0, 2\pi]$, the norm of the interpolation error must be at least $||\tau||$. We find the error in each coefficient by

$$\tilde{f}_k = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)e^{-ikx_j} = \frac{1}{N}\sum_{j=0}^{N-1}\left(\sum_{m=-\infty}^{\infty}\hat{f}_m e^{imx_j}\right)e^{-ikx_j} = \sum_{m=-\infty}^{\infty}\hat{f}_m\left(\frac{1}{N}\sum_{j=0}^{N-1}e^{i(m-k)x_j}\right) =$$

$$\sum_{m=-\infty}^{\infty}\hat{f}_m\left(\frac{1}{N}\sum_{j=0}^{N-1}\delta_{m-k,\pm pN}\right) = \sum_{p=-\infty}^{\infty}\hat{f}_{k+pN} = \hat{f}_k + \sum_{|p|=1}^{\infty}\hat{f}_{k+pN}. \tag{17}$$

One can show that $||P_N f - f|| \leq ||I_N f - f|| \leq 2||P_N f - f||$, see [3]. The difference $\tilde{f}_k - \hat{f}_k = \sum_{|p|\neq 0}^{\infty}\hat{f}_{k+pN}$ is called the *aliasing error*. Note however, that the aliasing error depends on how fast the Fourier coefficients decay.

4.3 Derivative of Interpolant

We approximate the derivative of the function by differentiating the interpolant

$$f'(x) \approx (I_N f)'(x) = \sum_{k=-N/2}^{N/2}\frac{ik\tilde{f}_k}{c_k}e^{ikx} \tag{18}$$

or, equivalently, differentiating the Langrange form

$$(I_N f)'(x) = \sum_{j=0}^{N-1} f(x_j)h_j'(x). \tag{19}$$

We have shown that truncation and differentiating commute, in (8). We will now show that interpolation and differentiation do not commute. We define the *remainder operator* R_N such that

$$I_N f = P_N f + R_N f$$

so that

$$R_N f = \sum_{k=-N/2}^{N/2}\sum_{|p|=1}^{\infty}\frac{\hat{f}_{k+pN}}{c_k}e^{ikx} - \frac{1}{2}\left(\hat{f}_{N/2}e^{iNx/2} + \hat{f}_{-N/2}e^{-iNx/2}\right).$$

We have that

$$(I_N f)' = (P_N f)' + (R_N f)' = P_N f' + (R_N f)'$$

while

$$I_N f' = P_N f' + R_N f'.$$

$(R_N f)' \neq R_N f'$, since

$$(R_N f)' = \sum_{k=-N/2}^{N/2}\sum_{|p|=1}^{\infty}\frac{ik\hat{f}_{k+pN}}{c_k}e^{ikx} - \frac{1}{2}\left(\frac{iN}{2}\hat{f}_{N/2}e^{iNx/2} - \frac{iN}{2}\hat{f}_{-N/2}e^{-iNx/2}\right)$$

and

$$R_N f' = R_N \left(\sum_{k=-N/2}^{N/2} \frac{i(k+pN)\hat{f}_k}{c_k} e^{ikx} \right) =$$

$$\sum_{k=-N/2}^{N/2} \sum_{|p|=1}^{\infty} \frac{i(k+pN)\hat{f}_{k+pN}}{c_k} e^{ikx} - \frac{1}{2} \left(\frac{iN}{2} \hat{f}_{N/2} e^{iNx/2} - \frac{iN}{2} \hat{f}_{-N/2} e^{-iNx/2} \right),$$

so the coefficients are not equal.

5 Lagrange Form of Fourier Interpolant

In the previous chapter we saw that one can write the interpolant in Lagrange form. By doing this, in this chapter we show that we can write the values of the function and its derivative at the nodes in closed form. This allows us to write the derivative of a periodic function at the nodes as a matrix-vector product. This will be how the method is implemented numerically. We finally show that the matrix in this matrix-vector product is skew-symmetric. The statements of theorems 14, 15 and 16 are from [2], although the proofs are derived by the author.

5.1 Rewriting the Lagrange Functions

We have shown that one can write the Fourier interpolant in Lagrange form

$$I_N f(x) = \sum_{j=0}^{N-1} f(x_j) h_j(x) \tag{20}$$

$$h_j(x) = \frac{1}{N} \sum_{k=-N/2}^{N/2} \frac{1}{c_k} e^{ik(x-x_j)} \tag{21}$$

$$c_k = \begin{cases} 1, & |k| \leq N/2 - 1 \\ 2, & |k| = N/2 \end{cases}. \tag{22}$$

We will now prove two theorems regarding writing the Lagrange functions (21) and their derivatives in closed form.

Theorem 14. *The Lagrange functions can be written in the closed form*

$$h_j(x) = \begin{cases} \frac{1}{N} \sin\left(\frac{N}{2}(x-x_j)\right) \cot\left(\frac{1}{2}(x-x_j)\right), & x \neq x_j = \frac{2\pi j}{N} \\ 1, & x = x_j \end{cases}$$

Proof. We define $\xi = \frac{1}{2}(x - x_j)$. Multiplying (21) by N, we get

$$Nh_j(\xi) = \sum_{k=-N/2}^{N/2} \frac{1}{c_k} e^{2ik\xi} = \sum_{n=0}^{N} \frac{1}{c_{n-N/2}} e^{2i\xi(n-N/2)} = e^{-iN\xi} \sum_{n=0}^{N} \frac{1}{c_{n-N/2}} e^{2in\xi} =$$

$$e^{-iN\xi} \left(\frac{1}{2} + \frac{1}{2} e^{2iN\xi} + \sum_{n=1}^{N-1} \left(e^{2i\xi}\right)^n \right).$$

Since $\sum_{n=1}^{N-1} \left(e^{2i\xi}\right)^n$ is a geometric sum, we get

$$\sum_{n=1}^{N-1} \left(e^{2i\xi}\right)^n = -1 + \sum_{n=0}^{N-1} \left(e^{2i\xi}\right)^n = \frac{1 - e^{2iN\xi}}{1 - e^{2i\xi}} - 1.$$

Then

$$e^{-iN\xi} \left(\frac{1}{2} + \frac{1}{2} e^{2iN\xi} + \frac{1 - e^{2iN\xi}}{1 - e^{2i\xi}} - 1 \right) = e^{-iN\xi} \left(\frac{1}{2} \left(e^{2iN\xi} - 1\right) + \frac{1 - e^{2iN\xi}}{1 - e^{2i\xi}} \right) =$$

$$e^{-iN\xi} \left(e^{2iN\xi} - 1\right) \left(\frac{1}{2} - \frac{1}{1 - e^{2i\xi}} \right) = \left(e^{iN\xi} - e^{-iN\xi}\right) \frac{1 - e^{2i\xi} - 2}{2(1 - e^{2i\xi})} =$$

$$\frac{e^{iN\xi} - e^{-iN\xi}}{2} \frac{e^{2i\xi} + 1}{e^{2i\xi} - 1} = \frac{e^{iN\xi} - e^{-iN\xi}}{2} \frac{e^{i\xi} + e^{-i\xi}}{e^{i\xi} - e^{i\xi}} = \frac{e^{iN\xi} - e^{-iN\xi}}{2i} \frac{2i}{e^{i\xi} - e^{-i\xi}} \frac{e^{i\xi} + e^{-i\xi}}{2}.$$

Using Euler's formulas

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}, \quad \sin(x) = \frac{e^{ix} - e^{-ix}}{2i},$$

we get

$$\frac{e^{iN\xi}-e^{-iN\xi}}{2i}\frac{2i}{e^{i\xi}-e^{-i\xi}}\frac{e^{i\xi}+e^{-i\xi}}{2}=\sin(N\xi)\frac{1}{\sin(\xi)}\cos(\xi)=\sin(N\xi)\cot(\xi)=$$

$$=\sin\left(\frac{N}{2}(x-x_j)\right)\cot\left(\frac{1}{2}(x-x_j)\right),$$

where $\cot(x)=\cos(x)/\sin(x)$. □

Theorem 15. *The derivative of the Lagrange functions evaluated at the nodes is given by*

$$h_j'(x_n)=\begin{cases}\frac{1}{2}(-1)^{n+j}\cot\left(\frac{(n-j)\pi}{N}\right), & n\neq j\\ 0, & n=j\end{cases}.$$

Proof. Again defining $\xi=\frac{1}{2}(x-x_j)$, we get

$$h_j(\xi)=\frac{1}{N}\sin(N\xi)\cot(\xi)\Rightarrow h_j'(\xi)=\cos(N\xi)\cot(\xi)-\frac{1}{N}\frac{\sin(N\xi)}{\sin^2(\xi)}\iff$$

$$\iff h_j'(x)=\frac{1}{2}\left(\cos\left(\frac{N}{2}(x-xj)\right)\cot\left(\frac{1}{2}(x-x_j)\right)-\frac{1}{N}\frac{\sin\left(\frac{N}{2}(x-x_j)\right)}{\sin^2\left(\frac{1}{2}(x-x_j)\right)}\right).$$

Taking the limit as $x\to x_n=\frac{2\pi n}{N}$ we get

$$\lim_{x\to\frac{2\pi n}{N}}h_j'(x)=\lim_{t\to1}\frac{1}{2}\left(\cos(\pi(tn-j))\cot\left(\frac{\pi}{N}(tn-j)\right)-\frac{1}{N}\frac{\sin(\pi(tn-j))}{\sin^2\left(\frac{\pi}{N}(tn-j)\right)}\right)=$$

$$\lim_{t\to1}\frac{1}{2}\left(\frac{\cos(\pi(tn-j))\cos\left(\frac{\pi}{N}(tn-j)\right)}{\sin\left(\frac{\pi}{N}(tn-j)\right)}-\frac{1}{N}\frac{\sin(\pi(tn-j))}{\sin^2\left(\frac{\pi}{N}(tn-j)\right)}\right)=$$

$$\lim_{t\to1}\frac{1}{2}\frac{\cos(\pi(tn-j))\frac{1}{2}\sin\left(\frac{2\pi}{N}(tn-j)\right)-\frac{1}{N}\sin(\pi(tn-j))}{\sin^2\left(\frac{\pi}{N}(tn-j)\right)}.$$

If $n\neq j$, since $0<|n-j|<N$ we get

$$h_j'(x_n)=\frac{1}{2}\frac{\cos(\pi(n-j))\frac{1}{2}\sin\left(\frac{2\pi}{N}(n-j)\right)-\frac{1}{N}\sin(\pi(n-j))}{\sin^2\left(\frac{\pi}{N}(n-j)\right)}=$$

$$\frac{1}{2}\frac{(-1)^{n-j}\sin\left(\frac{\pi}{N}(n-j)\right)\cos\left(\frac{\pi}{N}(n-j)\right)}{\sin^2\left(\frac{\pi}{N}(n-j)\right)}=\frac{1}{2}(-1)^{n+j}\cot\left(\frac{\pi}{N}(n-j)\right).$$

If $n=j$ then

$$\lim_{t\to1}\frac{1}{2}\frac{\cos(\pi(tn-j))\frac{1}{2}\sin\left(\frac{2\pi}{N}(tn-j)\right)-\frac{1}{N}\sin(\pi(tn-j))}{\sin^2\left(\frac{\pi}{N}(tn-j)\right)}=$$

$$\lim_{r\to0}\frac{1}{2}\frac{\cos(\pi r)\frac{1}{2}\sin\left(\frac{2\pi}{N}r\right)-\frac{1}{N}\sin(\pi r)}{\sin^2\left(\frac{\pi}{N}r\right)}=\lim_{r\to0}\frac{\frac{1}{2}\sin\left(\frac{2\pi}{N}\right)-\frac{1}{N}\sin(\pi r)}{1-\cos\left(\frac{2\pi}{N}r\right)}=$$

$$\lim_{r\to0}\frac{\frac{\pi}{N}\cos\left(\frac{2\pi}{N}r\right)-\frac{\pi}{N}\cos(\pi r)}{\frac{2\pi}{N}\sin\left(\frac{2\pi}{N}r\right)}=\lim_{r\to0}\frac{-\frac{2\pi^2}{N^2}\sin\left(\frac{2\pi r}{N}\right)+\frac{\pi^2}{N}\sin(\pi r)}{\frac{4\pi^2}{N}\cos\left(\frac{2\pi}{N}r\right)}=0,$$

where we used L'Hopital's rule for the 3'rd to last and 2'nd to last equalities. □

5.2 Lagrange Derivative Matrix

To see how an approximate solution written in Lagrange form $\varphi(x,t) = \sum_{j=0}^{N-1} \varphi(x_j,t)h_j(x)$ to (1) evolves over time, we only need to know how the node values $\varphi(x_j,t)$ evolve over time. We note that using (19) we can approximate the space derivative of f evaluated at the nodes by

$$\bar{f}'(t) \approx D\bar{f}(t),$$

where we define the *Lagrange derivative matrix* as $D_{i,j} = h_j'(x_i)$, i.e.

$$D = \begin{pmatrix} h_0'(x_0) & h_1'(x_0) & ... & h_{N-1}'(x_0) \\ h_0'(x_1) & h_1'(x_1) & ... & h_{N-1}'(x_1) \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ h_0'(x_{N-1}) & h_1'(x_{N-1}) & ... & h_{N-1}'(x_{N-1}) \end{pmatrix}. \tag{23}$$

Theorem 16. *The Lagrange derivative matrix D has eigenvalues $\lambda_k = ik$, $k = -\frac{N}{2} + 1, ..., \frac{N}{2} - 1$.*

Proof. D has elements $D_{i,j} = h_j'(x_i)$. We first note that $h_j'(x_j) = 0$, so all diagonal elements vanish. We secondly note that, if $j \neq i$

$$D_{i+1,j+1} = h_{j+1}'(x_{i+1}) = (-1)^{i+j+2}\frac{1}{2}\cot\left(\frac{((j+1)-(i+1))\pi}{N}\right) =$$

$$(-1)^{i+j}\frac{1}{2}\cot\left(\frac{(j-i)\pi}{N}\right) = D_{i,j}.$$

We thirdly note that since $\cot(\pi - x) = \cot(x)$ that $h_0'(x_2) = h_{N-1}'(x_1)$ etc. These three facts show that D is the circulant matrix $\text{circ}\{h_0'(x_0), ..., h_{N-1}'(x_0)\}$. Since $D \in \mathbb{C}^{N\times N}$ and $e^{2\pi i/N}$ is a primitive N'th root of unity, we get from theorem 7 that D has eigenvalues

$$\lambda_l = \sum_{j=0}^{N-1} e^{2\pi ijl/N} h_j'(x_0) = \sum_{j=0}^{N-1} e^{ilx_j} h_j'(x_0), \quad l = 0, 1, ..., N-1.$$

Identifying equations (18) and (19) we get that for 2π-periodic functions

$$\sum_{j=0}^{N-1} f(x_j)h_j'(x) = \sum_{k=-N/2}^{N/2} \frac{ik\tilde{f}_k}{c_k} e^{ikx}. \tag{24}$$

The functions $\{e_l\}_{l=0}^{n-1}$ are 2π-periodic and their discrete Fourier coefficients are

$$\tilde{e}_{lk} = \begin{cases} 1, & k = l \\ 0, & k \neq l \end{cases}.$$

Using equation (24) with $f = e_l$ at the point $x_0 = 0$, we get

$$\sum_{j=0}^{N-1} e^{ilx_j} h_j'(x_0) = \sum_{k=-N/2}^{N/2} \frac{ik\tilde{e}_{lk}}{c_k} = \frac{il}{c_k}, \quad l = 0, 1, ..., N/2 - 1.$$

If $l = N/2$, the $\tilde{e}_{l\,N/2} = \tilde{e}_{l-N/2} = 1$, so

$$\sum_{j=0}^{N-1} e^{ilx_j} h_j'(x_0) = \frac{iN/2}{2} + \frac{-iN/2}{2} = 0.$$

If $l = N/2 + 1, ..., N - 1$, then $\tilde{e}_{ll} = \tilde{e}_{ll-N}$, so

$$\sum_{j=0}^{N-1} e^{ilx_j} h_j'(x_0) = \frac{i(l-N)}{c_{l-N}} = i(l-N).$$

Altogether, these three cases show that D has eigenvalues

$$\lambda_k = ik, \quad k = -N/2 + 1, ..., N/2 - 1.$$

□

5.3 Symmetry of Discretization

When discretizing a differential operator it is of interest whether the discretization retains certain properties of the operator. An inner product space $(X, (\cdot, \cdot))$ is a linear space X together with a function $(\cdot, \cdot) : X \times X \to \mathbb{C}$ (called an inner product) satisfying

$$\begin{aligned} (x, y) &= (y, x)^\star && \text{(conjugate symmetry)} \\ (ax + by, z) &= a(x, z) + b(y, z),\ a, b \in \mathbb{C} && \text{(linearity in first argument)} \\ (x, x) &\geq 0,\ (x, x) = 0 \iff x = 0 && \text{(positive-definiteness).} \end{aligned}$$

From the conjugate symmetry and the linearity in the first argument we also get

$$(x, ay) = a^\star(x, y), \quad \text{(sesquilinearity).}$$

Definition 17. Let an operator A be defined on an inner product space $(X, (\cdot, \cdot))$. A is said to be *self-adjoint* if

$$(u, Av) = (Au, v), \quad \forall u, v \in X,$$

and *anti-self-adjoint* if

$$(u, Av) = -(Au, v), \quad \forall u, v \in X.$$

Theorem 18. *Self-adjoint operators have only real eigenvalues and anti-self-adjoint operators haver only purely imaginary eigenvalues.*

Proof. First, assume A is self-adjoint, and u is an eigenvector with eigenvalue λ. Then

$$\lambda ||u||^2 = (\lambda u, u) = (Au, u) = (u, Au) = (u, \lambda u) = \lambda^\star ||u||^2,$$

by the sesquilinearity of inner products, so $\lambda = \lambda^\star$, i.e. λ is real. Next, assume that A is anti-self-adjoint. Then

$$\lambda ||u||^2 = (\lambda u, u) = (Au, u) = -(u, Au) = -(u, \lambda u) = -\lambda^\star ||u||^2,$$

so $\lambda = -\lambda^\star$, i.e. λ is purely imaginary. □

Theorem 19. *The differential operators $\frac{\partial}{\partial x}$ and $\frac{\partial^2}{\partial x^2}$ defined on the inner product space $(C[0, 2\pi], (\cdot, \cdot))$ of continuous 2π-periodic functions along with the Fourier inner product are anti-self-adjoint and self-adjoint, respectively.*

Proof. Integrating by parts we get

$$\left(u, \frac{\partial v}{\partial x}\right) = \int_0^{2\pi} u(x) \frac{\partial v^\star(x)}{\partial x} dx = [u(x)v^\star(x)]_0^{2\pi} - \int_0^{2\pi} \frac{\partial u(x)}{\partial x} v^\star(x) dx = -\left(\frac{\partial u}{\partial x}, v\right),$$

and

$$\left(u, \frac{\partial^2 v}{\partial x^2}\right) = \int_0^{2\pi} u(x) \frac{\partial^2 v^\star(x)}{\partial x^2} dx = \left[u(x) \frac{\partial v^\star(x)}{\partial x}\right]_0^{2\pi} - \int_0^{2\pi} \frac{\partial u(x)}{\partial x} \frac{\partial v^\star(x)}{\partial x} dx =$$

$$= -\left[\frac{\partial u(x)}{\partial x} v^\star(x)\right]_0^{2\pi} + \int_0^{2\pi} \frac{\partial^2 u(x)}{\partial x^2} v^\star(x) dx = \left(\frac{\partial^2 u}{\partial x^2}, v\right).$$

□

The equivalent to an operator being self-adjoint or anti-self-adjoint for matrices is the matrix being symmetric or skew-symmetric. The following theorem shows that the matrices D and D^2 are skew-symmetric and symmetric, respectively.

Theorem 20. *The matrix D defined by (23) is skew symmetric, i.e. $D^T = -D$, and D^2 is symmetric, i.e. $(D^2)^T = D^2$.*

Proof. We compare the entries $D_{i,j}$ and $D_{j,i}$. If $j = i$, then $D_{i,j} = 0 = -D_{i,j}$. If $j \neq i$, then since the cot function is anti-symmetric

$$D_{j,i} = \frac{1}{2}(-1)^{i+j} \cot\left(\frac{(j-i)\pi}{N}\right) = -\frac{1}{2}(-1)^{j+i} \cot\left(\frac{(i-j)\pi}{N}\right) = -D_{i,j}.$$

Therefore, $D^T = -D$. Since for any matrix A, $(A^k)^T = (A^T)^k$, we have

$$(D^2)^T = (D^T)^2 = (-D)^2 = D^2.$$

□

D and D^2 will be used in the Fourier collocation method as discretizations of $\frac{\partial}{\partial x}$ and $\frac{\partial^2}{\partial x^2}$. That they retain the equivalent properties is therefore desirable. Note that we already knew that D has only purely imaginary eigenvalues by theorem 16.

6 The Fourier Collocation Method

In this chapter we first solve the advection-diffusion equation analytically. We then describe the Fourier collocation method and show some numerical results. We describe how to remove aliasing errors. The description of the method is based on [2], except for the details on how to remove the aliasing errors.

6.1 An Exact Solution

We will now find an exact solution to the advection-diffusion equation

$$\begin{cases} \varphi_t(x,t) + \varphi_x(x,t) = \nu\varphi_{xx}(x,t), & 0 < x < 2\pi \\ \varphi(x,0) = \varphi_0(x), & 0 \leq x \leq 2\pi \ , t > 0, \\ \varphi(0,t) = \varphi(2\pi,t) \end{cases} \tag{25}$$

where we set the initial function to the familiar function

$$\varphi_0(x) = b(x) = \frac{3}{5 - 4\cos(x)}.$$

As we have already mentioned, this function has the Fourier coefficients [2]

$$\hat{f}_k = 2^{-|k|},$$

so

$$\varphi_0(x) = \sum_{k=-\infty}^{\infty} 2^{-|k|} e^{ikx}.$$

We make the anzats

$$\varphi(x,t) = \sum_{k=-\infty}^{\infty} g_k(t) 2^{-|k|} e^{ikx}$$

and get the system of ordinary differential equations

$$\sum_{k=-\infty}^{\infty} \frac{dg_k}{dt} 2^{-|k|} e^{ikx} = \sum_{k=-\infty}^{\infty} -(\nu k^2 + ik) g_k(t) 2^{-|k|} e^{ikx} \iff \frac{dg_k}{dt}(t) = -(\nu k^2 + ik) g_k(t) \iff$$

$$g_k(t) = C_k e^{-(\nu k^2 + ik)t}.$$

The initial conditions give $g_k(0) = 1$, so the exact solution is

$$\varphi(x,t) = \sum_{k=-\infty}^{\infty} 2^{-|k|} e^{ik(x-t) - \nu k^2 t}.$$

This solution can now be used to test the accuracy of the numerical methods.

6.2 Describing the method

We are now ready to describe the Fourier collocation method. As a reminder, we want to find an approximate solution to the PDE

$$\begin{cases} \varphi_t(x,t) + \varphi_x(x,t) = \nu\varphi_{xx}(x,t), & 0 < x < 2\pi \\ \varphi(x,0) = \varphi_0(x), & 0 \leq x \leq 2\pi \ , t > 0. \\ \varphi(0,t) = \varphi(2\pi,t) \end{cases}$$

We wish to approximate the solution by a function ϕ defined on $x \in [0, 2\pi]$ and on the temporal grid points $t_n = n\Delta t$, where we have chosen the time step $\Delta t > 0$. We use the notation $\phi_k = \phi(t_k)$. We first choose an even number $N \geq 2$. To find an approximation we first create the vector $\bar{\phi}_0 = (\varphi_0(x_0), \varphi_0(x_1), ..., \varphi_0(x_{N-1}))^T$ (note the bar over the ϕ), where $x_j = \frac{2\pi j}{N}$, $j = 0, ..., N-1$. We then define the function ϕ_0 for all $x \in [0, 2\pi]$ by

$$\phi_0(x) = \sum_{j=0}^{N-1} \bar{\phi}_{0,j} h_j(x).$$

We now wish to approximate the time derivative at each node. We have from the PDE that

$$\varphi_t(x_j, 0) = \nu \varphi_{xx}(x_j, 0) - \varphi_x(x_j, 0).$$

We approximate the spatial derivative by differentiating the interpolant, written in Lagrange form

$$\varphi_x(x_n, 0) \approx \frac{d}{dx} \sum_{j=0}^{N-1} \bar{\phi}_0(x_j) h_j(x) \Big|_{x=x_n} = \sum_{j=0}^{N-1} \bar{\phi}_0(x_j) h_j'(x) \Big|_{x=x_n} = \sum_{j=0}^{N-1} D_{n,j} \phi_0(x_j),$$

where from theorem 15 we have that

$$D_{n,j} = h_j'(x_n) = \begin{cases} \frac{1}{2}(-1)^{n+j} \cot\left(\frac{(n-j)\pi}{N}\right), & n \neq j \\ 0, & n = j \end{cases}.$$

We create the matrix D by first calculating all off-diagonal values. To calculate the diagonal elements, we use the fact that the derivative of the function $f(x) = 1$ is identically zero and each node value is identically 1, so we should get $\sum_{j=0}^{N-1} D_{n,j} = 0$. We therefore set $D_{n,n} = -\sum_{j=0,\, j\neq n}^{N-1} D_{n,j}$. Due to round-off errors, the diagonal elements will not be exactly zero. As mentioned by [2], this method of calculating the diagonal elements makes the overall approximation of the derivative less susceptible to round-off errors. We can calculate the approximation of the spatial derivatives at all nodes simultaneously by setting

$$(\phi_x(x_0, 0), \phi_x(x_1, 0), ..., \phi_x(x_{N-1}, 0))^T \approx D(\phi(x_0, 0), \phi(x_1, 0), ..., \phi(x_{N-1}, 0))^T = D\bar{\phi}_0,$$

and likewise approximate the second spatial derivative at the nodes by

$$(\phi_{xx}(x_0, 0), \phi_{xx}(x_1), 0), ..., \phi_{xx}(x_{N-1}, 0))^T \approx D(\phi_x(x_0, 0), \phi_x(x_1, 0), ..., \phi_x(x_{N-1}, 0))^T = D^2\bar{\phi}_0.$$

By now viewing the equations describing the time derivative of each node value as a system of ODE's (and using the approximation of the spatial derivatives described above) we get the system of ODE's

$$\frac{d}{dt}\bar{\phi}_0 = -D(\bar{\phi}_0 - \nu D\bar{\phi}_0).$$

We then use some time stepping method, three different ones being described in the next chapter, to define the approximation $\bar{\phi}_1 = (\phi(x_0, \Delta t), \phi(x_1, \Delta t), ..., \phi(x_{N-1})\Delta t))^T$. We then continue like this, for as long as we like. Using e.g. the explicit Euler method, we get a recursive relation

$$\bar{\phi}_{k+1} = \bar{\phi}_k - \Delta t D(\bar{\phi}_k - \nu D\bar{\phi}_k).$$

Note that we do not need to generate D in every step, only one time, as the same matrix is being used each time. Note also that the next chapter does not mention the

explicit Euler method as a potential time stepping method. This is because not only is the method only order 1, but the region of stability is not suitable for the system of ODE's above. Once the vector $\bar{\phi_k}$ has been generated, we can define the approximation $\phi(x, k\Delta t)$ for any $x \in [0, 2\pi]$. We do this by writing ϕ in Lagrange form

$$\phi(x, k\Delta t) = \sum_{j=0}^{N-1} \bar{\phi}_{k,j} h_j(x),$$

where $\bar{\phi}_{k,j}$ is the j'th element of $\bar{\phi_k}$, starting at 0. Note that for each time t_k, function ϕ_k satisfies the advection-diffusion equation at the nodes x_j. A collocation method is a method which satisfies the differential equation at some specified points. This is the origin of "collocation" in the name "Fourier collocation method". Note however that this assumes that $(D^2\phi_k)_n = \frac{\partial^2}{\partial x^2}\phi(x_k, t_k)$. Since interpolation and differentiation do not commute, this is not entirely true, but an approximation. Therefore, the word "collocation" promises slightly more than is true.

The figures below show the some results from using the Fourier collocation method. Figure 1 shows the numerical solution for $\nu = 1/5$. Figure 2 shows the numerical solution for different times along with the reference solution (with Fourier modes up to $k = 1000$. For details on the time stepping used, see chapter 7. The initial function was $\varphi_0(x) = b(x)$, where $b(x)$ is the function in (13). These figures show that the method is qualitatively accurate, as not only does the initial function relax, but it also traverses in the positive x-direction, properties we expect of the solution. Figure 3 shows the relative error of the numerical solution for different non-positive values of ν. As one can see, the method performs very well for different values, and the relative error is largely equal to the initial error, which arises from the interpolation of the initial function. The discrete norm is spectrally accurate to the continuous norm, since they only differ for modes $|k| > N/2$.

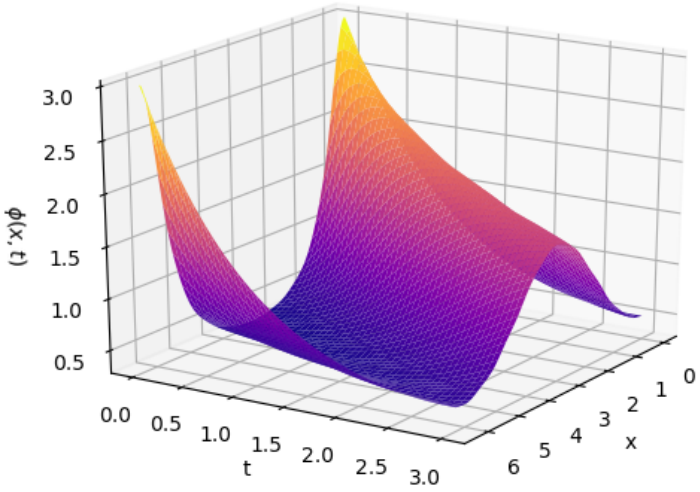


Figure 1. Solution of advection-diffusion equation with $\nu = 1/5$ with initial function $b(x)$ in (13). The number of modes used was $N = 16$, the time stepping was performed using Williamson's Runge-Kutta method and the time step was $\Delta t = 1.25 \times 10^{-3}$.

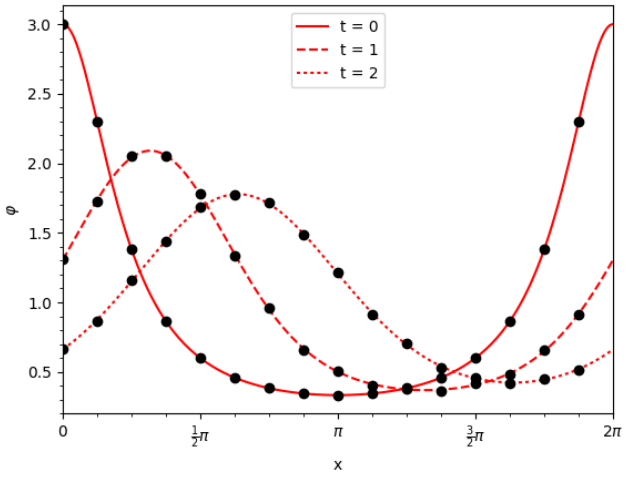


Figure 2. Approximated solution (circles) and exact solution (lines) for different times, with $\Delta t = 1.25 \times 10^{-3}$, $\nu = 1/5$. Williamson's method was used for time-stepping and the exact solution was calculated using Fourier modes up to $k = 1000$.

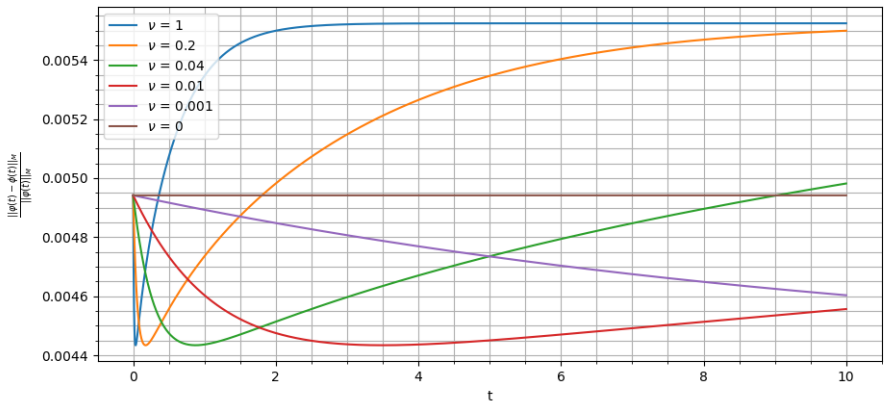


Figure 3. Relative error of solution as a function of time with different diffusion coefficients ν. The number of modes was $N = 16$, the time stepping was performed using the trapezoidal rule and the time step was $\Delta t = 1.25 \times 10^{-3}$. The discrete norm was measured using $M = 100$ points.

6.3 Removing Aliasing Errors

The Fourier collocation method starts by interpolating the initial conditions, and advances forward in time as though the initial function was replaced by its interpolant. The downside of doing this is that the interpolant $I_N\varphi_0$ is not the best approximation from φ_0 to $\mathcal{L}_{N/2}$ in the L_2 norm on $[0, 2\pi]$. The best approximation in this norm is the Fourier truncation $P_N\varphi_0$. Using the interpolant as initial function introduces what's called aliasing errors, as seen in (17). Suppose we instead use $P_N\varphi_0$ as initial condition. Remember that $P_N\varphi_0 = \sum_{k=-N/2}^{N/2} \hat{\varphi}_{0k} e_k$, where $\hat{\varphi}_{0k}$ are the Fourier coefficients

$$\hat{\varphi}_{0k} = \frac{1}{2\pi}(\varphi_0, e_k) = \frac{1}{2\pi}\int_0^{2\pi} \varphi_0(x) e^{ikx} dx.$$

The resulting initial function does not satisfy the initial condition at the nodes, leading to an initial error at the nodes. However, since the Fourier coefficients $\hat{f}_k$ decay as $e^{-\nu k^2}$, the coefficients of higher frequencies decay quickly, leading to fast convergence. The interpolant pools the higher frequency coefficients together with the lower frequency ones. This leads to the higher frequency coefficients decaying much slower than they are supposed to. Therefore, while using the Fourier coefficients leads to an initial error at the nodes, the error becomes much less than for the discrete coefficients, as time goes by.

As an example, let's use the case when $\varphi(x, 0) = \cos(2x) + \frac{1}{10}\cos(6x)$, and $N = 8$. We write the initial function as its Fourier series

$$\varphi(x, 0) = \cos(2x) + \frac{1}{10}\cos(6x) = \frac{1}{2}e^{2ix} + \frac{1}{2}e^{-2ix} + \frac{1}{20}e^{6ix} + \frac{1}{20}e^{-6ix},$$

so the non-zero Fourier coefficients are $\hat{f}_{-6} = \hat{f}_6 = \frac{1}{20}$, $\hat{f}_{-2} = \hat{f}_2 = \frac{1}{2}$. Let's also, for convenience, disregard the advection term. The exact solution (as can be seen by equation (4)) is

$$\varphi(x, t) = e^{-4\nu t}\cos(2x) + \frac{1}{10}e^{-36\nu t}\cos(6x).$$

Let's first use the discrete coefficients, the "intuitive choice". Note that I_N can only use terms up to $|k| = 4$ for $N = 8$. Since $\tilde{f}_k$, used to define $I_N f$ by theorem 12, is calculated using (15), we get

$$\tilde{f}_k = (f, e_k)_N = \left(\frac{1}{2}e_2 + \frac{1}{2}e_{-2} + \frac{1}{20}e_6 + \frac{1}{20}e_{-6}, e_k\right)_N =$$
$$\left(\frac{1}{2}e_2, e_k\right)_N + \left(\frac{1}{2}e_{-2}, e_k\right)_N + \left(\frac{1}{20}e_6, e_k\right)_N + \left(\frac{1}{20}e_{-6}, e_k\right)_N =$$
$$\frac{1}{2}\delta_{k,2\pm 8p} + \frac{1}{2}\delta_{k,-2\pm 8p} + \frac{1}{20}\delta_{k,6\pm 8p}\frac{1}{20}\delta_{k,-6\pm 8p},\ k = -4, ..., 4,$$

where $\delta_{k,j\pm 8p} = \begin{cases} 1, & \text{if } k = j + 8p \text{ or } k = j - 8p \text{ for some } p \in \{0, 1, 2, ...,\} \\ 0, & else \end{cases}$. Therefore, we get $\tilde{f}_{-2} = \tilde{f}_2 = \frac{11}{20}$. The Fourier collocation method now treats the differential equation as though the initial function was $\phi_I(x, 0) = \frac{11}{20}e^{2ix} + \frac{11}{20}e^{-2ix} = \frac{11}{10}\cos(2x)$. The exact solution with this initial function would then be

$$\phi_I(x, t) = \frac{11}{10}e^{-4\nu t}\cos(2x)$$

which again can be seen from (4). The squared norm of the error as a function of time is then

$$||(\phi_I - \varphi)(t)||^2 = \int_0^{2\pi} \left|\frac{1}{10}e^{-4\nu t}\cos(2x) + \frac{1}{10}e^{-26\nu t}\cos(6x)\right|^2 dx = \frac{\pi}{100}e^{-8\nu t} + \frac{\pi}{100}e^{-72\nu t}.$$

Let's now instead use the Fourier coefficients $\hat{f}_k$. Since again the higher coefficients cannot be resolved, this gives the approximate solution

$$\phi_P(x, t) = e^{-4\nu t}\cos(2x).$$

Now the squared norm of the error is

$$||(\phi_P - \varphi)(\cdot, t)||^2 = \int_0^{2\pi} \left|\frac{1}{10}e^{-36\nu t}\cos(6x)\right|^2 dx = \frac{\pi}{100}e^{-72\nu t}.$$

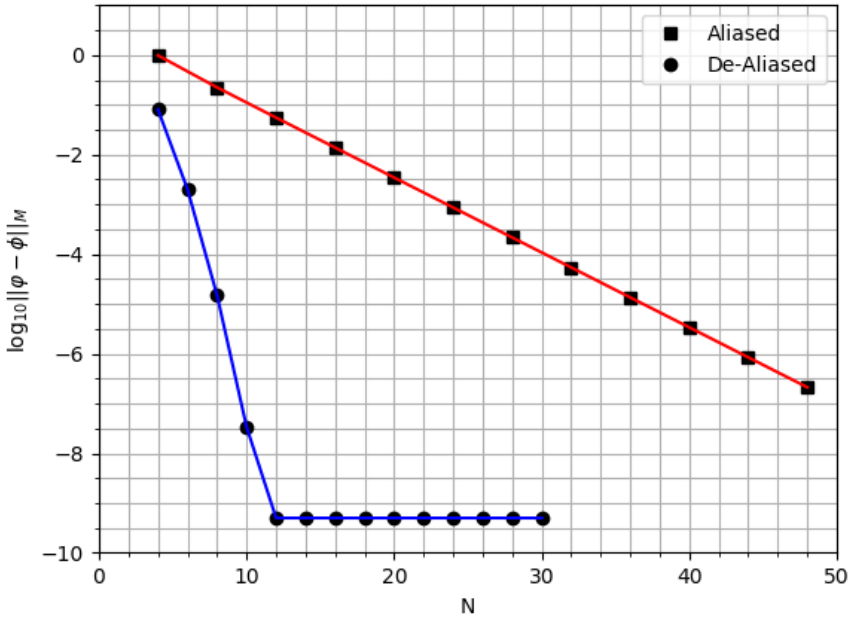
Not only is the error smaller for the truncation than for the interpolation, but the error also decays much faster. The one downside is that one has to first calculate the Fourier coefficients. Figure 4 and 5 show the error of the solution at $t = 3$ for different values of N. In figure 4 $\nu = 1/5$, and one can clearly see that the de-aliased solution has a much lower error. In figure 5 $\nu = 0$, and the error of the two initial functions is almost equal. This is explained by the fact that the smaller error of the de-aliased solution is based on the Fourier coefficients decaying fast. This is achieved when $\nu > 0$, since the coefficients are dampened by a factor of $e^{-k^2\nu t}$.

The problem with using de-aliased initial conditions is that one has to calculate the Fourier coefficients $\hat{\varphi}_0$. Doing this exactly requires evaluating an integral, which might not even be possible to evaluate analytically.Instead we calculate the discrete coefficients with a large number $N = N_{big}$, e.g. $N_{big} = 10N$. The Fourier coefficients then almost match the calculated discrete ones, since the aliasing error

$$\tilde{f}_k - \hat{f}_k = \sum_{|p|=1}^{\infty} \hat{f}_{pN_{big}}$$

becomes extremely small if the Fourier coefficients decay quickly, say with spectral speed. Since P_N and I_N are both projectors onto $\mathcal{L}_{N/2}$, $I_N(P_N f)) = P_N f$. Therefore, when

the Fourier collocation method interpolates the initial condition, if the initial function has been replaced by its Fourier truncation, then the coefficients will remain $\tilde{f}_k = \hat{f}_k$. Hence, nothing else is needed to remove aliasing errors but to replace the initial function by its Fourier truncation.



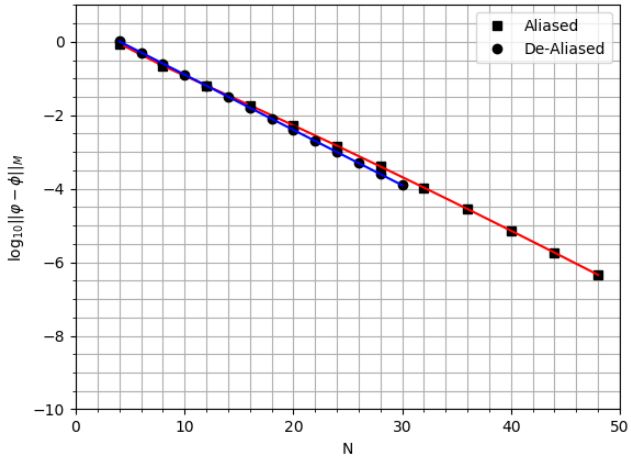


Figure 4 and 5. Error at time $t = 3$ for aliased (interpolation) and de-aliased (truncation) initial conditions as a function of N. The parameters were $\Delta t = 1.25 \times 10^{-3}$, $\nu = 1/5$. Williamson's method was used for time-stepping and the exact solution was calculated using Fourier modes up to $k = 1000$. Figure 4 (top) uses $\nu = 1/5$ and figure 5 (bottom) uses $\nu = 0$.

7 Time stepping

In this section we describe two time-stepping schemes with different pros and cons. We first describe Williamson's Runge-Kutta method, which is an explicit third order method. We then present the trapezoidal rule, which is an A-stable, implicit, second order method. Since the method involves inverting a specific matrix, we show that this is possible. We also show how one can use the structure of the matrices to integrate continuously in time. We use the example of wanting to solve the ODE

$$\begin{cases} u_t = f(t,u),\ t > 0 \\ u(0) = g \end{cases}$$

and discretizing the ODE on the time steps $u(t_n) \approx u^n$. The description of Williamson's Runge-Kutta method is from [2]. Everything in section 7.2 except the description of the trapezoidal rule and everything in section 7.3 is derived by the author.

7.1 Williamson's Method

Williamson's Runge-Kutta method is a third order explicit method [2]. It has the Butcher tableau

0			
1/3	1/3		
3/4	−3/16	15/16	
	1/6	3/10	8/15

The calculation can be done more storage efficiently by the following algorithm

$$\begin{aligned} u &\leftarrow u^n \\ g &\leftarrow f(u, t_n) \\ u &\leftarrow u + \tfrac{\Delta t}{3} g \\ g &\leftarrow -\tfrac{5}{9} g + f(t_n + \tfrac{\Delta t}{3}, u) \\ u &\leftarrow u + \tfrac{15\Delta t}{16} g \\ g &\leftarrow -\tfrac{153}{128} g + f(t_n + \tfrac{3\Delta t}{4}, u) \\ u^{n+1} &\leftarrow u + \tfrac{8\Delta t}{15} g. \end{aligned}$$

Note that for the Fourier collocation semi-discretization of the advection-diffusion equation, $f(t,u) = -D(I - \nu D)u$. The stability region can be seen in figure 6 below. Since the method is not A-stable, i.e. the stability region does not contain the left half-plane $\mathbb{C}^-$. This puts stability-restrictions on the time step Δt.

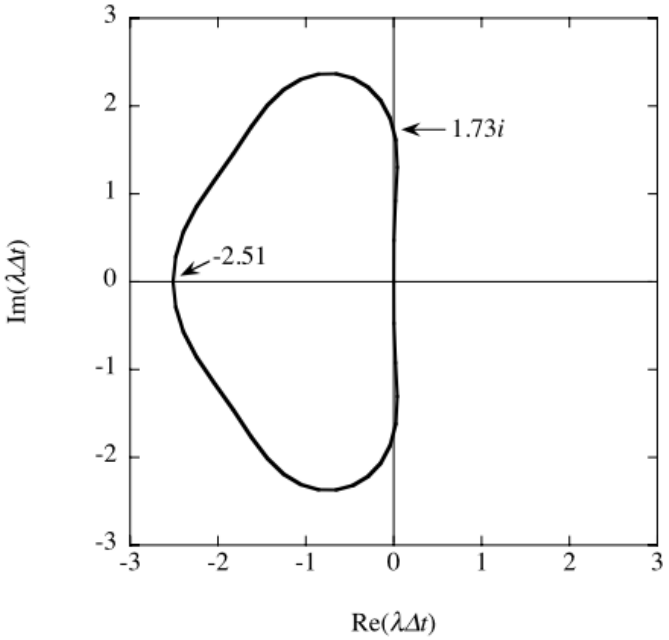


Figure 6. Stability region of Williamson's Runge-Kutta method. Figure is taken from figure 4.1 of [2].

7.2 Trapezoidal Rule

The trapezoidal rule is a second order implicit method. It uses the approximation

$$f(t_n, u(t_n)) \approx \frac{1}{2}\left(f(t_n, u^n) + f(t_{n+1}, u^{n+1}\right),$$

leading to the time stepping scheme

$$u^{n+1} = u^n + \frac{\Delta t}{2}\left(f(t_n, u^n) + f(t_{n+1}, u^{n+1})\right).$$

For the Fourier collocation semi-discretization of the advection-diffusion equation, again $f(t,u) = -D(I - \nu D)u$. Therefore, the equation can be re-written as

$$\left(I + \frac{\Delta t}{2} D(I - \nu D)\right) u^{n+1} = \left(I - \frac{\Delta t}{2} D(I - \nu D)\right) u^n.$$

Since the eigenvalues of $-D(I - \nu D)$ are $-(ik + \nu k^2)$, $k = -N/2 + 1, ..., N/2 - 1$ (see section 5.2), the left-hand matrix does not have any zero eigenvalue, and can therefore be inverted. The system of equations can easily be solved as below, using the fact that $\left(I + \frac{\Delta t}{2} D(I - \nu D)\right)$ is circulant

$$\begin{aligned} w = &\quad \left(I - \tfrac{\Delta t}{2} D(I - \nu D)\right) u^n \\ \Psi = &\quad U \left(I + \tfrac{\Delta t}{2} D(I - \nu D)\right) U^\star \\ u^{n+1} = &\quad U^\star \Psi^{-1} U w, \end{aligned}$$

since Ψ is a diagonal matrix, and its diagonal elements are the inverses of the diagonal entries of Ψ. The stability region of the trapezoidal rule is exactly the left half-plane $\mathbb{C}^-$, so there is no stability restriction on the time step. To be specific, we take a time

step using the trapezoidal rule in the following way:
Generate the matrix $U^\star$ with entries $U^\star_{kj} = e^{2\pi ikj/N}/\sqrt{N}$. Store this matrix and use it for every time step. Then, for each time step, do the following:

1. Set $A = -D(I - \nu D)$
2. $w = (I + \frac{\Delta t}{2}A)u_n$
3. $\Psi = U^\star(I - \frac{\Delta t}{2}A)U$
4. $w = Uw$
5. for $k = 1, 2, ..., N$ do:
   $w_k = w_k/\Psi_{kk}$
6. $u_{n+1} = U^\star w$.

Note that for the advection-diffusion equation, the matrix A can also be stored and reused, as well as Ψ. For other PDE's however, A might change for the different time steps, and so then will Ψ.

7.3 Continuous Evolution

We note that the advection-diffusion equation is a linear PDE, and that the spatially discretized approximation

$$\bar{u}_t = -D(I - \nu D)\bar{u}$$

is a system of linear ordinary differential equation. We also note that since D and I are both circulant matrices, the matrix $-D(I - \nu D)$ is also circulant. Therefore, as seen in chapter 3, the system of ODE's is easily solvable exactly, and is given by the equation

$$\begin{aligned} \bar{u}(t) = &\quad U^\star e^{t\Psi} U \bar{u}_0 \\ U_{j,k} = &\quad \tfrac{1}{\sqrt{N}} e^{-2\pi ijk/N} \\ \Psi = &\quad U(-D(I - \nu D))U^\star. \end{aligned}$$

Note again that Ψ is diagonal, and e^Ψ is the exponential of the entries. One major inconvenience with this time-stepping is that it only works for circulant linear systems of ODE's, and so only of periodic linear PDE's. The major benefit of the method is that not only does it get rid of time stepping errors completely, but one can also quickly compute the solution for any time t, not only the time steps t_n.

Figure 7 shows the error of the numerical solution for different time steps Δt, using the trapezoidal rule and Williamson's method. One can see that the trapezoidal rule is order 2 and Williamson's method is order 3. This is true up to a minimum error. Figure 8 shows the error of the numerical solution for the three different time stepping methods, with aliased and de-aliased initial conditions. The three methods have the same slope, the difference being the minimum error. We can see that the minimum error of Williamson's method is $\sim 10^{-9}$ while the trapezoidal rule has a minimum error $\sim 10^{-6}$. The ratio of exponents seem to be approximately 3/2, which reminds one of the time stepping orders of the methods. The minimum error of the continuous evolution is $\sim 10^{-13}$ and is fluctuating noticeably. The size of the error along with the fluctuations suggests that this error comes from round-off errors. We can also see that de-aliasing the

initial condition increases the speed of convergence, not the minimum error, although the increased speed heavily depends on the diffusion constant ν.

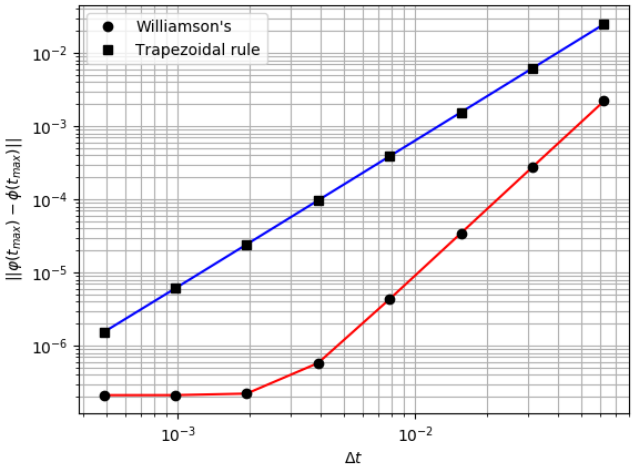


Figure 7. Error of solution at $t_{max} = 10$ for the advection-diffusion equation with $\nu = 1/5$ using $N = 48$. The time stepping was performed using the trapezoidal rule and Williamson's method. The error was calculated at $M = 100$ points, and the reference solution was calculated with $N = 500$, so modes up to $k = 250$. The initial function was $b(x)$ in (13), and the initial error was calculated using this function (i.e. exactly).

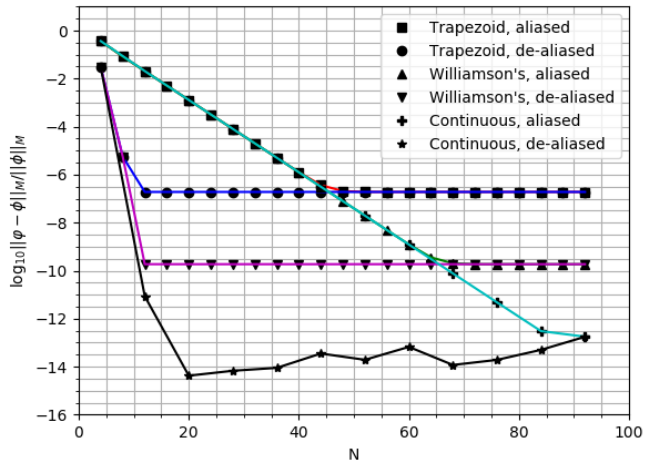


Figure 8. Relative error at time $t = 3$ for aliased (interpolation) and de-aliased (truncation) initial conditions, using Williamson's method, the trapezoidal rule and continuous time-stepping, as a function of N. The parameters were $\Delta t = 1.25 \times 10^{-3}$, $\nu = 1/5$. The reference solution was calculated using Fourier modes up to $k = 1000$, and the initial function was $b(x)$ in (13).

8 Other Initial Functions

In the previous chapters we have focused on the function $b(x) = 3/(5 - 4\cos(x))$. This is because its periodic extension is infinitely smooth, i.e. $b^P \in C^\infty(\mathbb{R})$, and we know its Fourier coefficients $\hat{b}_k = 2^{-|k|}$. One might then ask if the phenomena we have found generalize to other initial functions? We consider five additional functions whose periodic extensions are of varying smoothness. The functions, along with their Fourier coefficients are

$$p(x) = x(2\pi - x), \qquad \hat{p}_k = \begin{cases} \frac{2\pi^2}{3}, & k = 0 \\ -\frac{2}{k^2}, & k \neq 0 \end{cases}$$

$$c(x) = x(x - \pi)(x - 2\pi), \qquad \hat{c}_k = \begin{cases} 0, & k = 0 \\ \text{sign}(k)\frac{6i}{k^3}, & k \neq 0 \end{cases}$$

$$q(x) = x^2(x - 2\pi)^2, \qquad \hat{q}_k = \begin{cases} \frac{8\pi}{15}, & k = 0 \\ -\frac{24}{k^4}, & k \neq 0 \end{cases}$$

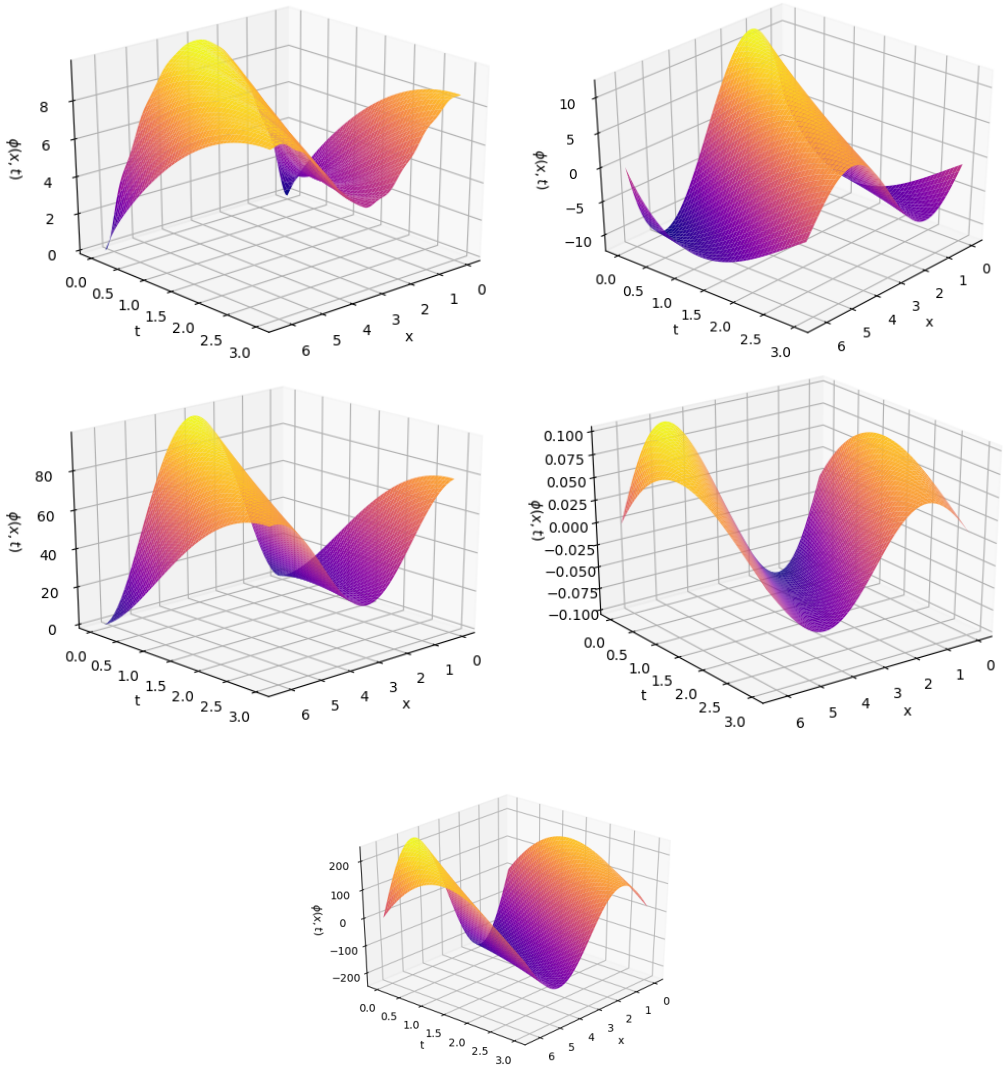
$$f(x) = x^2(x - \pi)(x - 2\pi)^2 - \frac{4\pi^2}{3}x(x - \pi)(x - 2\pi), \qquad \hat{f}_k = \begin{cases} 0, & k = 0 \\ -\text{sign}(k)\frac{120i}{k^5}, & k \neq 0 \end{cases}$$

$$s(x) = \sum_{|k|=1}^{\infty} 2i \cdot \text{sign}(k) 10^{-|k|} e^{ikx}, \quad \hat{s}_k = \begin{cases} 0, & k = 0 \\ 2i \cdot \text{sign}(k) 10^{-|k|}, & k \neq 0 \end{cases}.$$

The periodic extensions are of following smoothness

$$\begin{aligned} p^P &\in C(\mathbb{R}) \\ c^P &\in C^1(\mathbb{R}) \\ q^P &\in C^2(\mathbb{R}) \\ f^P &\in C^3(\mathbb{R}) \\ s^P &\in C^\infty(\mathbb{R}). \end{aligned}$$

The impact of the smoothness of the periodic extensions can be seen in the accuracy of the Fourier collocation method and the order of the time stepping methods. Figures 9-12 show the numerical solutions of the advection-diffusion equation with the different initial functions. This is to show what the numerical solutions look like, and does not give any direct insight into the accuracy of the methods. It is however worth to note that in figure 9, the numerical solution starts with a cusp, i.e. a point where the derivative switches sign discontinuously. The solutions is then smoothened out, as the method cannot represent cusps in the interior points $(0, 2\pi)$. Note that at a cusp, the second order space derivative, and therefore also the time derivative, is infinite.

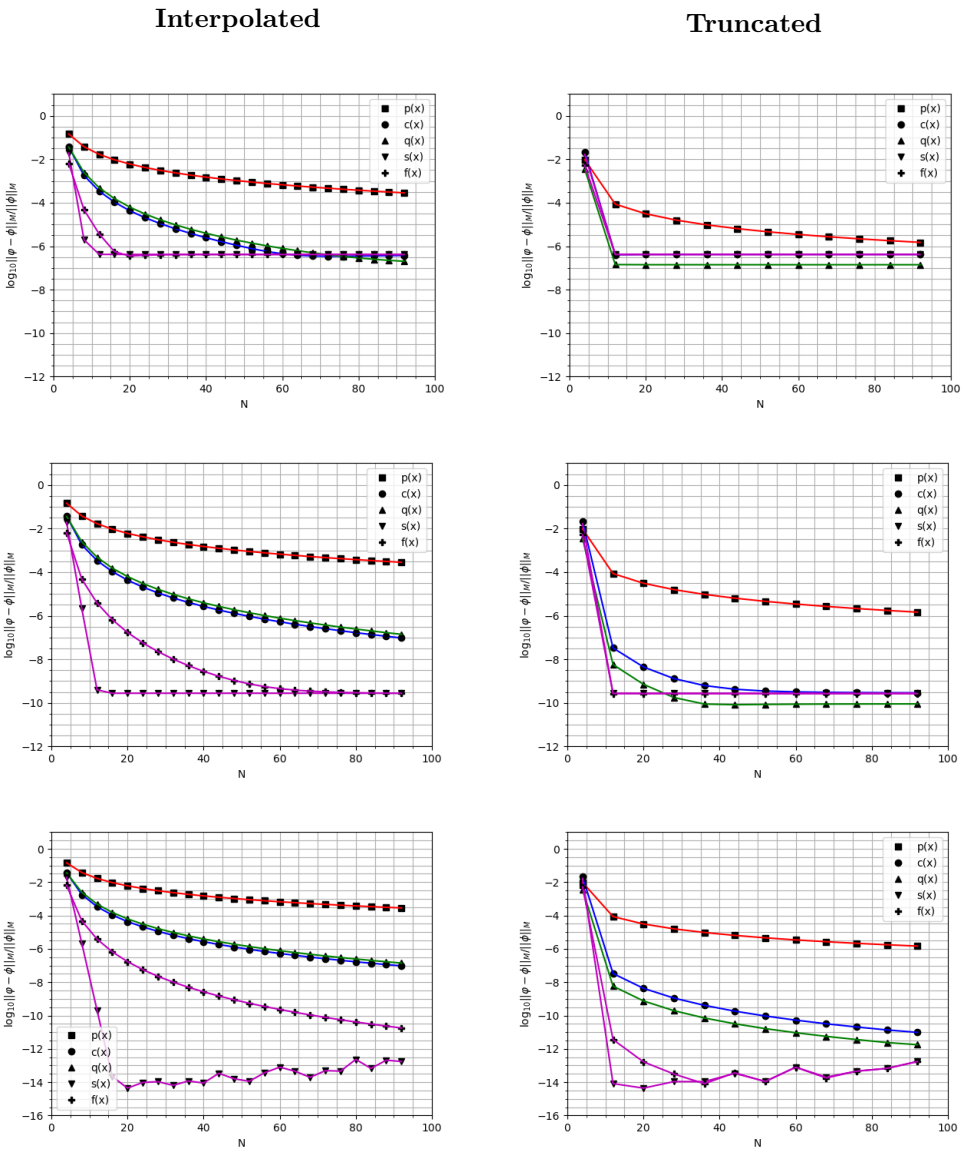


Figures 9-13. Numerical solution of advection-diffusion equation with $\nu = 1/5$. The number of modes used was $N = 16$, the time stepping was performed using Williamson's Runge-Kutta method and the time step was $\Delta t = 1.25 \times 10^{-3}$. The error was calculated at $M = 100$ points. Figure 9 (top left) uses initial function $p(x)$, figure 10 (top right) uses initial function $c(x)$, figure 11 (middle left) uses initial function $q(x)$, figure 12 (middle right) uses initial function $s(x)$ and figure 13 (bottom) uses initial function $f(x)$.

Figures 14-19 show the relative error of the numerical solution for different number of nodes N. Each figure shows the error for the four different initial functions, using some time stepping method. The left figures use interpolated initial functions and the right figures use truncated initial functions. In all figures we can see that the error decreases with increasing N faster for the smoother functions. This comes from the fact that the Fourier coefficients decay faster for smoother functions. Note that there is a smallest possible error, where the data flattens out.

From figures 14-17 we note (by looking at the data for $c(x)$, $q(x)$, $f(x)$ and $s(x)$) that using truncated initial functions does not lower this minimal error, but the error reach this level for lower values of N. Comparing figures 14, and 17 with figures 15

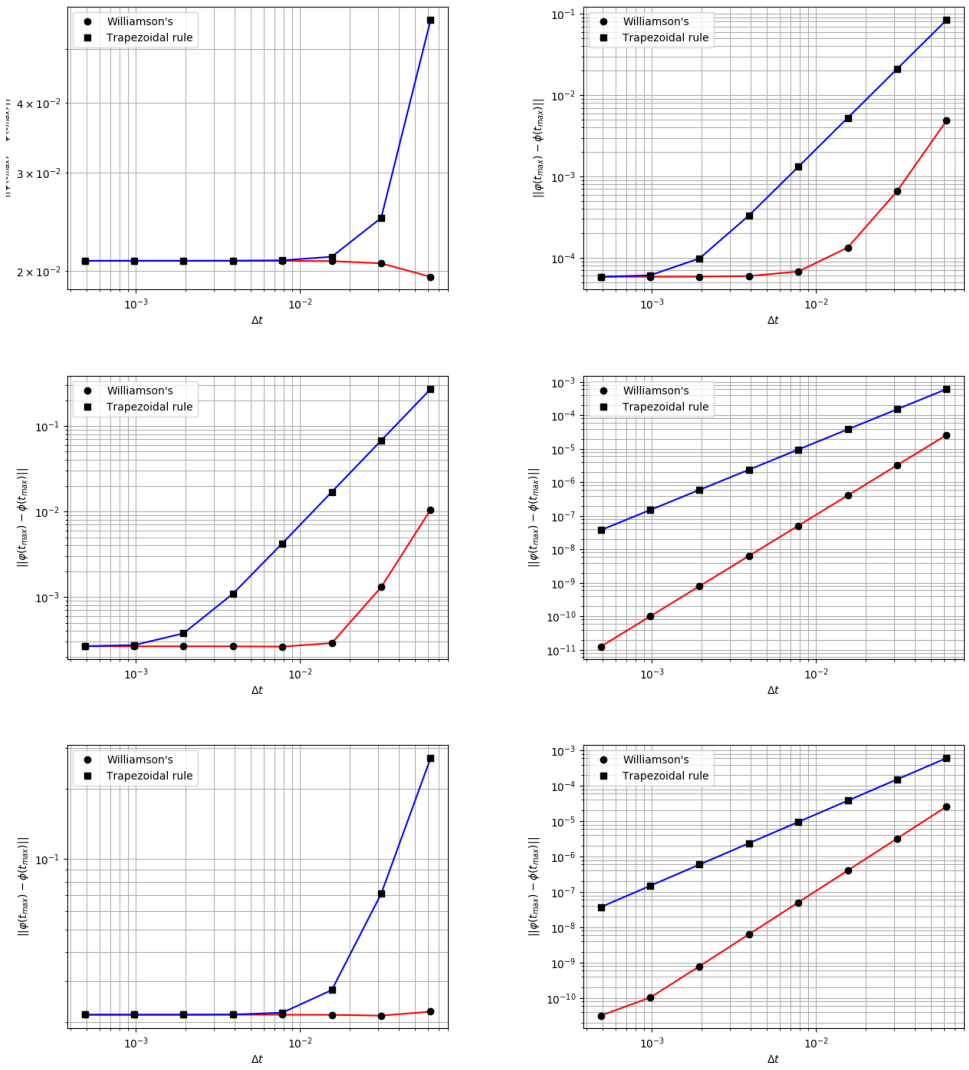
and 17 we see that using the third order Williamson's Runge-Kutta method reduces the minimal error compared with using the second order trapezoidal rule, at least for the functions $c(x)$, $q(x)$, $f(x)$ and $s(x)$. By comparing figures 16 and 18 with figures 17 and 19 shows that using continuous evolution instead of Williamson's method removes the barrier altogether, it seems. Note however that this does not apply to $p(x)$. This probably has to do with the periodic extension p^P being only continuous and piecewise C^1, so the initial function has a discontinuity in its first spatial derivative. Note also that the error decreases with increasing N at approximately the same rate for $c(x)$ and $q(x)$, although they have different minimal errors. Why this is is unknown to the author.



Figures 14-19. Error of solution at $t_{max} = 3$ for the advection-diffusion equation with $\nu = 1/5$ and $\Delta t = 1.25 \cdot 10^{-3}$ using different values of N. Each figure displays the error for the five different initial functions $p(x)$, $c(x)$, $q(x)$, $f(x)$ and $s(x)$. Figures 14 and 15 (he top two figures) used trapezoidal rule for time stepping, figures 16 and 17 (the middle two figures) used Williamson's Runge-Kutta method and figures 18 and 19 (the bottom two figures) used continuous evolution. The left figures used interpolated initial functions and the right figures used truncated initial functions. The error was calculated at $M = 100$ points, and the reference solutions were calculated with $N = 2000$, so modes up to $k = 1000$.

Figures 20-21 show the error of the solution for different time steps Δt, with time stepping performed using Williamson's Runge-Kutta and the trapezoidal rule. It is known that Williamson's Runge-Kutta method is order 3 and the trapezoidal rule is order 2. Therefore, since the axes are log-log, the slopes should be 3 and 2, respectively. We see that this holds for all figures up to a minimal error (except for figure 23 where the

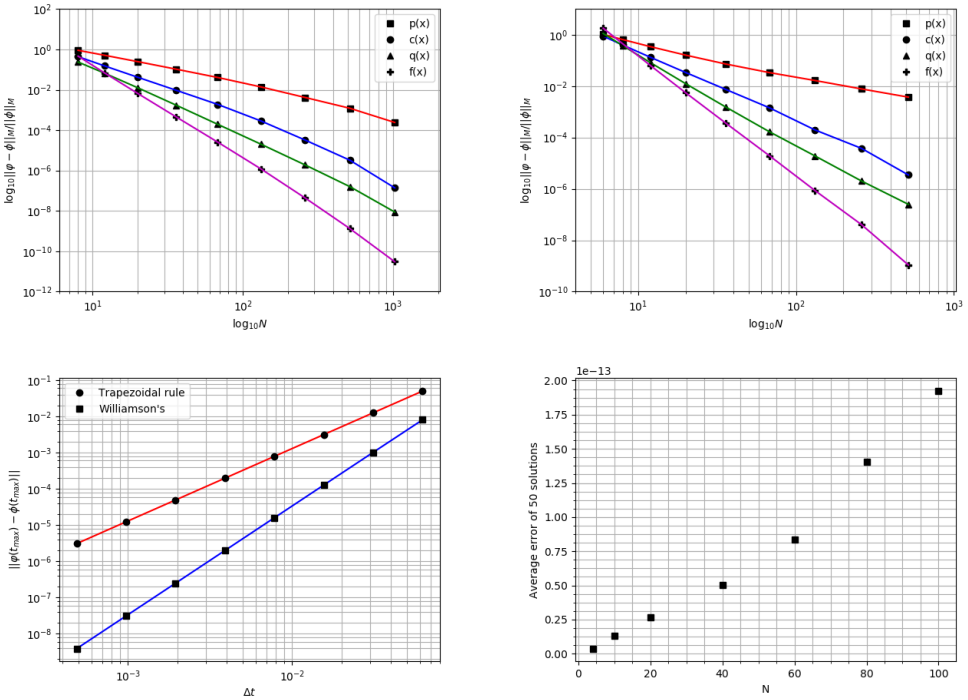
minial error has not yet been reached). This minimal error is smaller for the smoother functions. In figure 20, where the initial function is $p(x)$, this error is reached so quickly that it is hard to even see if it is ever order 2 for this initial function. If we compare figure 22 and 24, the only difference is the number of nodes N. We can see that for $N = 16$ (figure 24), the minimal error is much larger than for $N = 48$ (figure 22). Comparing figures 24 and 25, the only difference is that figure 24 uses interpolated initial functions and figure 25 uses truncated initial functions. We can see that the minimal error is much smaller for the truncated initial functions than for the interpolated initial functions.



Figures 20-25. Error of solution at $t_{max} = 10$ for the advection-diffusion equation with $\nu = 1/100$ using a fixed value of N and different times steps Δt. In each figure the error is calculated using Williamson's Runge-Kutta and the trapezoidal rule. The error is calculated at $M = 100$ points and the reference solution is calculated using $N = 500$, so modes up to $k = 250$. All figures except the bottom right use interpolated initial functions. Figures 20-23 (the top four) use $N = 48$ with initial functions $p(x)$ (top left), $c(x)$ (top right), $q(x)$ (middle left) and $s(x)$ (middle right). Figure 23 (bottom left) uses initial function $q(x)$ and $N = 16$. Figure 24 (bottom right) uses truncated initial function $q(x)$ and $N = 16$.

Figures 14-25 show the interplay between the two sources of error. There is the discretization error, coming from the fact that we are approximating the solution at time t with a finite series expansion in terms of Fourier basis functions. The second error is the time stepping error, coming from approximating the solution to the ODE arising from the semi-discretization of the PDE. Both errors set a level for the minimal error possible. This means that reducing the time step or increasing N only lowers the error up to a point, determined by the other variable. Note however than since the continuous evolution method supposedly solves the ODE exactly, there shouldn't be any time stepping error. Therefore, one would expect that the solutions shouldn't have any minimal error, except for the one arising from round-off errors. By studying figure 19, we see that this seems to be true.

Finally, figures 26-29 show errors from isolated parts of the method. Figure 26 shows how the relative interpolation error for different initial functions, decreases with N. Figure 27 shows on the other hand shows the relative truncation error. Both these show that the relative error decreases as N^{-n}, for some n. In fact, the convergence rate for the function whose Fourier coefficients decay as $1/k^p$ is approximately N^{-p}, as predicted by equation (10). Figure 28 shows the error as a function of time step Δt. The difference between figure 27 and figures 20-25 however, is that the reference solution is figures 20-25 was the calculated reference solution of the PDE. In figure 27 the reference solution is the solution of the ODE that arises from the discretization. In this way we have completely removed the discretization error, and the minimal error present in figures 20-25 does not exist in figure 27. This shows that the trapezoidal rule is in fact order 2, and Williamson's Runge-Kutta method is in fact order 3. Finally, for figure 29 the continuous evolution method was applied to 50 random $N \times N$ circulant matrices for various N. The average relative error is plotted. Even though the average relative error grows with increasing N, the error is in the order 10^{-13}, which is very small. This shows that the continuous evolution method works as intended.



Figures 26-29. Figure 26 (top left) and figure 27 (top right) shows the relative error of the initial Fourier interpolant and Fourier truncation, respectively, for different N. Figure 28 (bottom left) shows the error of the solution of the advection-diffusion equation at $t_{max} = 3$, where time stepping was performed using the trapezoidal rule and Williamspn's Runge-Kutta. The number of nodes used was $N = 16$, and the reference solution was the solution to the system of ODE's arising from the discretization, calculated using Williamson's Runge-Kutta with $\Delta t = 10^{-5}$. Figure 29 (bottom right) shows the average relative error of the solution to $x_t = Ax$, for random circulant matrices A, using continuous evolution. For each N, 50 solutions were calculated at $t_{max} = 3$, and the average relative error was plotted.

9 Summary and Further Topics

In this thesis we have reviewed some theory of Fourier series and Fourier polynomials, and how Fourier series can theoretically be used to solve PDE's with periodic boundary conditions. We have presented and compared two ways of approximating a function by a finite sum of Fourier basis functions, Fourier interpolation and Fourier truncation. We have also described how one can approximate the spatial derivative of a Fourier interpolant, and how the approximation evaluated at the nodes can be described as a matrix-vector product. We showed that the matrix in this product was circulant, and we showed that all circulant matrices have an easily computable diagonalization. We also showed that the matrix was skew-symmetric, and that the matrix used to approximate the second order spatial derivative was symmetric.

We described the Fourier collocation method in detail, and described how one implements it for solving the advection-diffusion equation with periodic boundary conditions, up to a choice of time stepping method for evolving in time. We described what aliasing errors are, how they arise and how one can remove them by first replacing the initial function with a Fourier truncation.

We showed that using the fact that all matrices involved in the semi-discretization of the PDE are circulant, this allowed us to advance the system in time without time stepping error. This however, only works since the advection-diffusion equation is linear, does not explicitly depend on time and we imposed periodic boundary conditions. We also described two other time stepping methods, namely Williamson's Runge-Kutta method and the trapezoidal rule. Since the trapezoidal rule involves inverting matrices, we proved that this is possible and showed how one can do it using the diagonalization. Finally, we presented some numerical results using various initial functions, diffusion constants, time step sizes and time stepping methods.

The numerical results allowed us to make some conclusions about the Fourier collocation method. The error of the solution decreases both with increasing number of nodes N and with decreasing time step size Δt, although increasing N or decreasing Δt only lowers the error down to a minimal error. If one wishes to lower the error further, one has to decrease Δt or increase N, respectively. One can also use a higher order time stepping method. The accuracy of the method is dependent on the speed of decay of the Fourier coefficients, and therefore the smoothness of the initial function. However, while the Fourier coefficients of $c(x)$ decay as $1/k^3$ and the Fourier coefficients of $q(x)$ decay as $1/k^4$, the error of both decrease equally fast when increasing N, which is troubling, as functions whose Fourier coefficients decay faster should be approximated more accurately using a fixed value of N. The method is designed for PDE's with periodic boundary conditions, so setting the initial function to $p(x) = x(2\pi - x)$, whose periodic extension is not C^1 made the results above regarding how the error decreases not hold entirely.

We demonstrated that the method works well for different diffusion constants ν, even for $\nu = 0$, when the equation reduces down to the advection equation, as can be seen in figure 3. It has also been demonstrated that unless $\nu = 0$, removing the initial aliasing error of the initial function leads to a great improvement in accuracy.

There are many possibilities for further studies of the subject area. It would be nice to have quantitative error estimates for the temporal and spatial discretizations, so that one could optimize the use of resources. I.e. one does not want to use more nodes or smaller step sizes than necessary, as this increases the computational effort without

reducing the error. The advection-diffusion equation can also be extended to use a non-constant diffusion constant $\nu(x)$, which is common in real life applications. It is also useful to extend the method to multiple space dimensions, as most applications are in at least 2D. Finally, the Fourier collocation method is designed for periodic boundary conditions. If one uses another basis of functions, e.g. Chebyshev polynomials, one can solve PDE's without periodic boundary conditions. Although one should point out that a lot of the results underlying the Fourier collocation method hold specifically for the Fourier basis functions, so one would need to design the method differently.

If the reader would like to read up on the subject of the Fourier collocation method, it is worth pointing out that the method heavily uses the discrete Fourier transform (DFT). A lot of the theory and description of the method could have been formulated in terms of the DFT, although the author made the choice not to do so. E.g., the discrete coefficients $\hat{f}_k$ defined in chapter 5 are defined using the inverse DFT.

A Python Code

A.1 Functions Used in the Simulation Methods

```
def nodes(N):
    """ Calculates the Fourier nodes xj = 2pi/N*j, j = 0, ..., N-1"""
    return np.array(np.linspace(0,2*np.pi,N+1)[:N])

def DFT(fj):
    """ Calculates the Discrete Fourier Transform based on the node values"""
    N = len(fj)
    Fk = []
    K = [k for k in range(-int(N/2), int(N/2))]
    for k in K:
        s = 0 + 0j
        for n in range(N):
            #s = s + fj[n]*(np.cos(-2*np.pi*n*k/N) + 1j*np.sin(-2*np.pi*n*k/N))
            s = s + fj[n]*np.e**(-2j*np.pi*n*k/N)
        Fk.append(s/N)
    return Fk

def modesToNodes(Fk):
    """ Calculates the node values based on the discrete Fourier coefficients"""
    N = len(Fk)
    fj = [0+0j]*len(Fk)
    for j in range(N):
        s = 0 + 0j
        for k in range(N):
            s = s + Fk[k]*np.e**(2j*np.pi*j*(k-N/2)/N)
        fj[j] = s
    return fj

def discreteTruncate(Fk,N):
    """ Takes the Fourier coefficients Fk[k] and returns then if -N/2 <= k <=
        N/2-1"""
    Nbig = len(Fk)
    for k in range(Nbig):
        #if abs(k-int(Nbig/2)) >= N/3+1:
        if abs(k-int(Nbig/2)) >= int(N/2):
            Fk[k] = 0
    #print(int(np.ceil(N/3)))
    return Fk[int(Nbig/2) - int(N/2):int(Nbig/2) + int(N/2)]

def deAliasedStart(N, Nbig, f0):
    """ Calculates the de-aliased initial condition, by truncating Nbig Fourier
    coefficients down to N."""
    Xj = nodes(Nbig)
    fj = f0(Xj)
    Fk = DFT(fj)
    Fk = discreteTruncate(Fk,N)
    fj_dealiased = modesToNodes(Fk)
    return fj_dealiased


def FourierDerivativeMatrix(N):
    """ Computes the Fourier interpolation derivative matrix basen on number of
```

```
    nodes N, even positive number"""
    D = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            if i!=j:
                D[i,j] = 0.5*(-1)**(i+j)/np.tan((i-j)*np.pi/N)
                D[i,i] -= D[i,j]
    return D


def UMatrix(N):
    """ Calculates the unitary matrix U in the eigenvalue decomposition of a
        circulant matrix """
    U = np.zeros((N,N), dtype = complex)
    for i in range(N):
        for k in range(N):
            U[i,k] = np.e**(2j*np.pi*i*k/N)/np.sqrt(N)
    return np.matrix(U)

def MatrixExp(A):
    """ Calculates the matrix exponential of a diagonal matrix """
    N = np.shape(A)[0]
    E = np.zeros((N,N), dtype = complex)
    for i in range(N):
        E[i,i] = np.e**(A[i,i])
    return np.matrix(E)


def discreteNorm(u):
    """ Computes the discrete norm of a vector, specrally accurate"""
    N = len(u)
    s = 0
    for i in range(N):
        s = s + 1/N*abs(u[i])**2
    return np.sqrt(2*np.pi*s)

def almostEqual(a,b):
    """ Used in the"""
    epsilon = 1.e-14

    if a == 0 or b == 0:
        if abs(a-b) <= 2*epsilon:
            return True
        else:
            return False
    else:
        if abs(a-b) <= epsilon*abs(a) and abs(a-b) <= epsilon*abs(b):
            return True
        else:
            return False

def interpolantFromNodes(x, fj):
    """ Computes the Fourier interpolant based on the node values"""
    N = len(fj)
    xj = nodes(N)
    for j in range(N):
        if almostEqual(x,xj[j]):
```

```
        return fj[j]
    if almostEqual(x,2*np.pi):
        return fj[0]

    s = 0
    for k in range(N):
        t = (x-xj[k])/2
        s = s+fj[k]*np.sin(N*t)/(np.tan(t)*N)
    return s
```

A.2 Initial Functions and Time Stepping Methods

```
""" The functions ending in _T calculate the solution of the
    advection-diffusion equation with the corresponding initial function, at
    the time t = T, spatial point x and diffusion constant d. The solution is
    calculated using Fourier coefficients up to N/2."""

def b(x):
    return 3/(5-4*np.cos(x))

def b_T(N,d,x,T):

    s = 1
    for k in range(1,int(N/2)):
        s = s + np.e**(-d*k**2*T)*np.cos(k*(x-T))*2**(-(k-1))
    return s

def p(x):
    return x*(2*np.pi - x)

def p_T(N,d,x,T):
    s = 2/3*np.pi**2 + 0j
    for k in range(1,int(N/2)+1):
        s = s - np.e**(-d*k**2*T)*np.cos(k*(x-T))*4/(k**2)
    return s

def c(x):
    return x*(x-np.pi)*(x-2*np.pi)

def c_T(N,d,x,T):
    s = 0+0j
    for k in range(1,int(N/2)+1):
        s = s + np.e**(-d*k**2*T)*np.sin(k*(x-T))*12/(k**3)
    return s

def q(x):
    return x**2*(x-2*np.pi)**2

def q_T(N,d,x,T):
    s = 16/5*np.pi**4-8*np.pi**4+16/3*np.pi**4 + 0j
    for k in range(1,int(N/2)+1):
        s = s - np.e**(-d*k**2*T)*np.cos(k*(x-T))*48/(k**4)
    return s

def f(x):
    return x**2*(x-2*np.pi)**2*(x-np.pi) -4/3*np.pi**2*x*(x-np.pi)*(x-2*np.pi)

def f_T(N,d,x,T):
    s = 0
    for k in range(1,int(N/2)+1):
        s = s - np.e**(-d*k**2*T)*np.sin(k*(x-T))*240/(k**5)
    return s

def s(x):
    s = 0 + 0j
    for k in range(1,501):
```

```
        s = s - np.sin(k*x)*10**(-k)
    return s

def s_T(N,d,x,T):
    s = 0+0j
    for k in range(1,int(N/2)+1):
        s = s - np.e**(-d*k**2*T)*np.sin(k*(x-T))*10**(-k)
    return s


    """ Different time stepping methods """

def trapezoidstep(uold, A, dt, U):
    """ Trapezoidal rule """
    n = max(np.shape(uold))
    uold = np.array(uold).reshape((n,1))
    I = np.eye(n, dtype = complex)
    w = (I+dt/2*A)@uold
    Psi = U@(I-dt/2*A)@U.H
    w = U@w
    for k in range(n):
        w[k,0] /= Psi[k,k]
    unew = U.H@w
    return np.array([unew[k,0] for k in range(n)])

def RKWstep(uold,A,dt, U):
    """ Williamson's Runge-Kutta method """
    u = uold
    G = A@uold
    u = u + dt*G/3
    G = -5/9*G + A@u
    u = u + 15/16*G*dt
    G = -153/128*G + A@u
    unew = u + 8/15*dt*G
    return unew
```

A.3 Simulation methods

```
def spectralAdvectionDiffusion_int(d,f0,tmax,dt,N, iterator):
    """ Approximately solves the advection-diffusion equation using the
    Fourier collocation method and aliased initial conditions"""
    xj = nodes(N)
    D = FourierDerivativeMatrix(N)

    M = int(tmax/dt)+1
    Solution = np.zeros((M,N), dtype = complex)
    Solution[0] = f0(xj)

    I = np.eye(N, dtype = complex)
    U = UMatrix(N)
    A = -D@(I-d*D)

    for i in range(1,M):
        Solution[i] = iterator(Solution[i-1], A, dt, U)

    return Solution

def spectralAdvectionDiffusion_dealiased(d,f0,tmax,dt,N, iterator):
    """ Approximately solves the advection-diffusion equation using the
    Fourier collocation method and de-aliased initial conditions"""
    D = FourierDerivativeMatrix(N)

    M = int(tmax/dt)+1
    Solution = np.zeros((M,N))
    Solution[0] = deAliasedStart(N, 10*N, f0)

    I = np.eye(N, dtype = complex)
    U = UMatrix(N)
    A = -D@(I-d*D)

    for i in range(1,M):
        Solution[i] = iterator(Solution[i-1], A, dt, U)

    return Solution

def sADError(d,tmax,dt,nstart,Nstep,nmax,Nbig,iterator,start,true, rel = False):
    """ Calculates the error as a function of N, with aliased initial
    condition."""
    errorvector = np.zeros(nmax+1)
    Nvector = [nstart + Nstep*k for k in range(nmax+1)]
    X = nodes(100)
    for k in range(nmax+1):
        N = Nvector[k]
        xj = nodes(N)
        D = FourierDerivativeMatrix(N)
        M = int(tmax/dt)+1

        fj = start(xj)
        I = np.eye(N, dtype = complex)
        U = UMatrix(N)
        A = -D@(I-d*D)
```

```
        for i in range(1,M):
            fj = iterator(fj, A, dt,U)

        solution_continuous = np.array([interpolantFromNodes(x, fj) for x in X])
        truesol = np.array([true(Nbig,d,x,tmax) for x in X])
        difference = solution_continuous - truesol
        if rel:
            errorvector[k] = discreteNorm(difference)/discreteNorm(truesol)
        else:
            errorvector[k] = discreteNorm(difference)
    return Nvector, list(errorvector)

def sADError_dealiased(d,tmax,dt,nstart,Nstep,nmax,Nbig,iterator,start,true,
    rel = False):
    """ Calculates the error as a function of N, with de-aliased initial
    condition."""
    errorvector = np.zeros(nmax+1)
    Nvector = [nstart + Nstep*k for k in range(nmax+1)]
    X = nodes(100)
    for k in range(nmax+1):
        N = Nvector[k]
        D = FourierDerivativeMatrix(N)
        M = int(tmax/dt)+1
        fj = deAliasedStart(N, 10*N, start)

        I = np.eye(N, dtype = complex)
        U = UMatrix(N)
        A = -D@(I-d*D)
        for i in range(1,M):
            fj = iterator(fj, A, dt, U)

        solution_continuous = np.array([interpolantFromNodes(x, fj) for x in X])
        truesol = np.array([true(Nbig,d,x,tmax) for x in X])
        difference = solution_continuous - truesol
        if rel:
            errorvector[k] = discreteNorm(difference)/discreteNorm(truesol)
        else:
            errorvector[k] = discreteNorm(difference)
    return Nvector, list(errorvector)

def sADError_cont(d,tmax,nstart,Nstep,nmax,Nbig,start,true,rel = False):
    """ Calculates the error as a function of N, with aliased initial
    condition and continuous evolution """
    errorvector = np.zeros(nmax+1)
    Nvector = [nstart + Nstep*k for k in range(nmax+1)]
    X = nodes(2000)
    Normvector = []
    for k in range(nmax+1):
        N = Nvector[k]
        xj = nodes(N)
        D = FourierDerivativeMatrix(N)
        I = np.eye(N)
        fj = np.transpose(np.matrix(start(xj)))

        A = -D@(I-d*D)
        U = UMatrix(N)
        Psi = U@A@U.H
```

```
        E = MatrixExp(tmax*Psi)
        fj = U@fj
        fj = E@fj
        fj = U.H@fj
        fj = np.transpose(fj)
        fj = np.resize(fj, N)

        solution_continuous = np.array([interpolantFromNodes(x, fj) for x in X])

        Normvector.append(discreteNorm(solution_continuous))
        truesol = np.array([true(Nbig,d,x,tmax) for x in X])

        difference = np.transpose(solution_continuous - truesol)
        if rel:
            errorvector[k] = discreteNorm(difference)/discreteNorm(truesol)
        else:
            errorvector[k] = discreteNorm(difference)
    return Nvector, list(errorvector)

def sADError_cont_dealiased(d,tmax,nstart,Nstep,nmax,Nbig,start,true,rel =
    False):
    """ Calculates the error as a function of N, with de-aliased initial
    condition and continuous evolution """
    errorvector = np.zeros(nmax+1)
    Nvector = [nstart + Nstep*k for k in range(nmax+1)]
    X = nodes(100)
    Normvector = []
    for k in range(nmax+1):
        N = Nvector[k]
        D = FourierDerivativeMatrix(N)
        I = np.eye(N)
        fj = np.transpose(np.matrix(deAliasedStart(N, 10*N, start)))

        A = -D@(I-d*D)
        U = UMatrix(N)
        Psi = U@A@U.H
        E = MatrixExp(tmax*Psi)
        fj = U@fj
        fj = E@fj
        fj = U.H@fj
        fj = np.transpose(fj)
        fj = np.resize(fj, N)

        solution_continuous = np.array([interpolantFromNodes(x, fj) for x in X])

        Normvector.append(discreteNorm(solution_continuous))
        truesol = np.array([true(Nbig,d,x,tmax) for x in X])

        difference = np.transpose(solution_continuous - truesol)

        if rel:
            errorvector[k] = discreteNorm(difference)/discreteNorm(truesol)
        else:
            errorvector[k] = discreteNorm(difference)
    return Nvector, list(errorvector)
```

References

[1] A. Holst, *Fourier Analysis*, Lund University Faculty of Science, Centre for Mathematical Sciences, Lund, 2014.

[2] D. A. Kopriva, *Implementing Spectral Methods for Partial Differential Equation, Algorithms for Scientists and Engineers*, Springer, 2009

[3] C. Canuto, M. Y. Hussaisi, A. Quarteroni, T. Zang, *Spectral Methods: Fundamentals in Single Domains*, Springer, Berlin, 2006

[4] http://www.math.stonybrook.edu/ sorin/eprints/circulant.pdf, last visited 2018-03-13.

[5] E. Stein, R. Shakararchi, *Fourier Analysis, An Introduction* Princeton University Press, Princeton and Oxford, 2003

[6] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations, Second Edition* Cambridge University Press, Cambridge, 2009