Numerical Solution of the Boundary Value Problems for Partial Differential Equations. Crash course for holographer

Alexander Krikun

Instituut-Lorentz, Universiteit Leiden, Delta-ITP
P.O. Box 9506, 2300 RA Leiden, The Netherlands

Abstract: These are the notes for a series of Numerical Study group meetings, held in Lorentz institute in the fall of 2017. The aim of the notes is to provide a non-specialist with the minimal knowledge in numerical methods used in BVP for PDEs, necessary to solve the problems typically arising in applications of holography to condensed matter systems. A graduate level knowledge of Linear Algebra and theory of Differential Equations is assumed. Special attention is payed to the treatment of the boundary conditions of general form. The notes focus on the practical aspects of the implementation leaving aside the theory behind the methods in use. A few simple problems to test the acquired knowledge are included.

Contents

1 Introduction

The applications of AdS/CFT to condensed matter physics have passed the stage of the "proof of concept" and as the problems under the focus become more involved, the more sophisticated numerical machinery is needed to track them. This sets a considerable obstacle to the community of holographers who, being trained as theoretical physicists, often lack the necessary numerical skills. The goal of these notes is to provide a detailed tutorial, to those willing to learn how to use numerical techniques in solving partial differential equations, which may arise in holographic problems, including, for instance holographic lattices [1–3].

The methods outlined here are applicable to the equations which may have singular points at the boundaries and to the problems with arbitrary boundary conditions. We intentionally avoid discussing any theory of Applied Mathematics, lying behind these methods since this is better explained in the excellent standard courses [4–6]. Most of the content here is in a big part copied from these books. Anyway, we tried to be more explicit when dealing with the systems of equations, and implementation of the boundary conditions. The subjects which are usually omitted in the standard courses, since they are very straightforward, but proved to be quite confusing when one is dealing with them for the first time.

2 Linear equations with constant coefficients

Consider a differential equation with constant coefficients (Internal Equation)

$$\text{IE:} \qquad A\,\partial_z^2 f(z) + B\,\partial_z f(z) + C\,f(z) = R \tag{2.1}$$

with boundary conditions (top and bottom)

$$\text{BCb:} \qquad B_b\,\partial_z f(z_b) + C_b\,f(z_b) = R_b \tag{2.2}$$

$$\text{BCt:} \qquad B_t\,\partial_z f(z_t) + C_t\,f(z_t) = R_t \tag{2.3}$$

The essence of Finite Difference Derivative (FDD) method is to turn this linear differential equation problem on the interval $z \in [z_b, z_t]$ into a system of linear algebraic equations. One does it by introducing the **grid** in the coordinate domain consisting of N points:

$$\vec{z} = \{z_1, z_2, \ldots, z_N\} \in [z_b, z_t], \qquad z_1 = z_b, \quad z_N = z_t \tag{2.4}$$

The simplest example would be

$$z_i = z_b + \frac{z_t - z_b}{N-1}(i-1), \qquad i = 1 \ldots N \tag{2.5}$$

Note that the boundary points **are included** in the grid.

$$z_1 \equiv z_b, \qquad z_N \equiv z_t \tag{2.6}$$

The function (f) and its derivatives $(\partial_z f, \partial_z^2 f)$ are naturally promoted to be the N**-component vectors**, representing the corresponding values at the grid points:

$$f(z) \to \vec{f} \qquad f_i \equiv f(z_i) \tag{2.7}$$

$$\partial_z f(z) \to \overrightarrow{\partial_z f} \qquad (\partial_z f)_i \equiv \partial_z f(z_i) \tag{2.8}$$

$$\partial_z^2 f(z) \to \overrightarrow{\partial_z^2 f} \qquad (\partial_z^2 f)_i \equiv \partial_z^2 f(z_i) \tag{2.9}$$

2.1 Differrentiation matrices

At this stage the question arises: Given the values of the function on the grid, how can one evaluate the values of its derivatives on the same grid? Firstly, note that because differentiation is a linear operation, the vectors $\overrightarrow{\partial_z f}$ and $\overrightarrow{f}$ are related by the linear transformation. Therefore the matrices exist, such that

$$\overrightarrow{\partial_z f} = \mathbb{D}_z \cdot \overrightarrow{f} \tag{2.10}$$

and similarly

$$\overrightarrow{\partial_z^2 f} = \mathbb{D}_{zz} \cdot \overrightarrow{f} \tag{2.11}$$

The $N \times N$ matrices $\mathbb{D}_z$ and $\mathbb{D}_{zz}$ are called **differentiation matrices**.

What is the explicit form of the differentiation matrix? Consider for example the simplest, *nearest neighbour* scheme for the first derivative matrix $\mathbb{D}_z$. The prescription for the finite difference derivative in this case is:

$$(\partial_z f)_i = \frac{f_{i+1} - f_{i-1}}{z_{i+1} - z_{i-1}} = \frac{f_{i+1} - f_{i-1}}{2\Delta z}, \qquad i \neq \{1, N\}. \tag{2.12}$$

Clearly, this formula is inapplicable at the boundaries. On these points one has to use **one-sided derivatives** instead:

$$(\partial_z f)_1 = \frac{-3f_1 + 4f_2 - f_3}{2\Delta z} \tag{2.13}$$

$$(\partial_z f)_N = \frac{f_N - 4f_{N-1} + 3f_{N-2}}{2\Delta z} \tag{2.14}$$

Combining these expressions together we can write down the explicit form of the differentiation matrix:

$$\mathbb{D}_z = \frac{1}{2\Delta z} \begin{pmatrix} -3 & 4 & -1 & \dots & 0 & 0 & 0 \\ -1 & 0 & 1 & \dots & 0 & 0 & 0 \\ 0 & -1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & -1 & 0 & 1 \\ 0 & 0 & 0 & \dots & 3 & -4 & 1 \end{pmatrix} \tag{2.15}$$

Similarly one constructs the matrix $\mathbb{D}_{zz}$.

Clearly, the nearest neighbour approximation is not the only possible choice for discretizing the derivative. One can use two, three or more nearest points in order to approximate the derivative with better accuracy. This leads to a denser differentiation matrix. For the details of the various approximations and their accuracy we refer the reader to the excellent tutorial "The Numerical Method of Lines" of Wolfram Mathematica [7].

2.2 Operator of the internal equations

Using the differentiation matrices, one can recast the differential equation (2.1) as a linear system

$$(A\,\mathbb{D}_{zz} + B\,\mathbb{D}_z + C\,\mathbb{I})\cdot\vec{f} - R\,\vec{1} = 0, \tag{2.16}$$

where $\mathbb{I}$ is $N\times N$ identity matrix and $\vec{1}$ is N vector of unities. We can introduce the **linear operator**

$$\mathbb{O} \equiv A\,\mathbb{D}_{zz} + B\,\mathbb{D}_z + C\,\mathbb{I}. \tag{2.17}$$

Then the vector of the equations (2.1) on the grid reads

$$\overrightarrow{IE}: \quad \mathbb{O}\cdot\vec{f} = R\,\vec{1} \tag{2.18}$$

2.3 Boundary conditions

It is important to note here that the linear system (2.18) **is not** equivalent to the boundary value problem (2.1), (2.2), since it does not include the boundary conditions yet. Indeed, $(\overrightarrow{IE})_1$ and $(\overrightarrow{IE})_N$ are the equations (2.1) evaluated on the endpoints, which we should substitute with the appropriate discretized boundary conditions (2.2). Moreover, in practice it often happens that the internal equations (2.1) are singular on the boundaries and evaluating them on the endpoints doesn't make any sense.

In order to discretize the boundary conditions (2.2) one can follow the same procedure as for the internal equations. The equations (2.2) (for all grid points z_i) can be represented as

$$\overrightarrow{BCb} = \mathbb{O}_b\cdot f - R_b\,\vec{1}, \qquad \overrightarrow{BCt} = \mathbb{O}_t\cdot f - R_t\,\vec{1} \tag{2.19}$$

$$\mathbb{O}_b \equiv B_b\,\mathbb{D}_z + C_b\,\mathbb{I} \qquad \mathbb{O}_t \equiv B_t\,\mathbb{D}_z + C_t\,\mathbb{I} \tag{2.20}$$

We do not actually need the boundary conditions at **all** points, since we should only keep $(\overrightarrow{BCb})_1$ for the condition at z_b and $(\overrightarrow{BCt})_N$ for the condition at z_t. More precisely, we need

$$(\overrightarrow{BCb})_1 = (\mathbb{O}_b)_{1j}(\vec{f})^j - R_b(\vec{1})_1, \tag{2.21}$$

$$(\overrightarrow{BCt})_N = (\mathbb{O}_t)_{Nj}(\vec{f})^j - R_t(\vec{1})_N, \tag{2.22}$$

where the summation over j is assumed and only the **first line** of $\mathbb{O}_b$ and **last line** of $\mathbb{O}_t$ are used.

2.4 Operator of the full problem

One can merge (2.18) and (2.21) in a single system of N linear equations, which represent the full boundary value problem (2.1), (2.2):

$$\overrightarrow{\text{BVP}}: \quad \begin{cases} (\overrightarrow{BC b})_1 \\ (\overrightarrow{IE})_2 \\ \dots \\ (\overrightarrow{IE})_{N-1} \\ (\overrightarrow{BC t})_N \end{cases} = \begin{cases} (\mathbb{O}_b)_{1j}(\vec{f})^j - R_b\,(\vec{1})_1 \\ (\mathbb{O})_{2j}(\vec{f})^j - R\,(\vec{1})_2 \\ \dots \\ (\mathbb{O})_{N-1\,j}(\vec{f})^j - R\,(\vec{1})_{N-1} \\ (\mathbb{O}_t)_{Nj}(\vec{f})^j - R_t\,(\vec{1})_N \end{cases} \equiv \tilde{\mathbb{O}} \cdot \vec{f} - \vec{\tilde{R}} \tag{2.23}$$

Here we introduced the **operator of the boundary value problem** (BVP operator) $\tilde{\mathbb{O}}$, which coincides with $\mathbb{O}$ everywhere except the first and the last lines, which are substituted from $\mathbb{O}_b$ and $\mathbb{O}_t$, respectively. Similarly, the vector of the right hand side $\vec{\tilde{R}}$ coincides with $R\,\vec{1}$ everywhere except first and last entry, where the values R_b and R_t are substituted.

In the end of the day, the BVP problem (2.1), (2.2) can be recast in the matrix form

$$\tilde{\mathbb{O}} \cdot \vec{f} - \vec{\tilde{R}} = 0, \tag{2.24}$$

and can be solved by direct inversion of the BVP operator

$$\vec{f} = \tilde{\mathbb{O}}^{-1} \cdot \vec{\tilde{R}} \tag{2.25}$$

2.5 Assignment

Solve the equation

$$f''(z) + \pi^2 f(z) = 0 \tag{2.26}$$

in the domain $z \in [0, 1]$ with the boundary conditions

$$f(0) = 1, \qquad f'(1) = 0 \tag{2.27}$$

Solution

$$f(z) = \cos(\pi z) \tag{2.28}$$

3 Linear equations with variable coefficients

In the previous section we have already encountered the situation where the values of the coefficients $\vec{R}$ in (2.24) are different inside the grid and on the endpoints. In this section we generalize this feature to the arbitrary variable coefficients on the grid.

Consider a linear differential equation with variable coefficients

$$\text{IE:} \qquad A(z)\,\partial_z^2 f(z) + B(z)\,\partial_z f(z) + C(z)\,f(z) = R(z) \tag{3.1}$$

with boundary conditions

$$\text{BCb:} \qquad B_b\,\partial_z f(z_b) + C_b\,f(z_b) = R_b \tag{3.2}$$

$$\text{BCt:} \qquad B_t\,\partial_z f(z_t) + C_t\,f(z_t) = R_t \tag{3.3}$$

3.1 Vectors of coefficients

Now not only the function and its derivatives will become vectors of values on the grid, but also the coefficients of the equation. Similarly to (2.7) we get:

$$X(z) \to \vec{X}, \qquad X_i \equiv X(z_i), \qquad X \in \{A, B, C, R\} \tag{3.4}$$

Since the coefficients take different values, one effectively has to solve *different algebraic equations at every grid point*. Using the same notation as (2.23), we write down the resulting system of equations equivalent to the boundary value problem.

$$\overrightarrow{\text{BVP}} = \begin{cases} (\overrightarrow{BCb})_1 \\ (\overrightarrow{IE})_2 \\ \dots \\ (\overrightarrow{IE})_{N-1} \\ (\overrightarrow{BCt})_N \end{cases} = \begin{cases} (0\,\mathbb{D}_{zz} + B_b\,\mathbb{D}_z + C_b\,\mathbb{I})_{1j}(\vec{f})^j - R_b \\ (A_2\,\mathbb{D}_{zz} + B_2\,\mathbb{D}_z + C_2\,\mathbb{I})_{2j}(\vec{f})^j - R_2 \\ \dots \\ (A_{N-1}\,\mathbb{D}_{zz} + B_{N-1}\,\mathbb{D}_z + C_{N-1}\,\mathbb{I})_{N-1\,j}(\vec{f})^j - R_{N-1} \\ (0\,\mathbb{D}_{zz} + B_t\,\mathbb{D}_z + C_t\,\mathbb{I})_{Nj}(\vec{f})^j - R_t \end{cases} \tag{3.5}$$

We can rewrite it as a matrix equation like (2.24)

$$\tilde{\mathbb{O}} \cdot \vec{f} - \vec{\tilde{R}} = 0 \tag{3.6}$$

by defining the linear operator matrix line-wise, i.e.

$$\tilde{\mathbb{O}}_{ij} \equiv \tilde{A}_i(\mathbb{D}_{zz})_{ij} + \tilde{B}_i(\mathbb{D}_z)_{ij} + \tilde{C}_i(\mathbb{I})_{ij}, \qquad i, j = 1 \dots N. \tag{3.7}$$

Or in the matrix form

$$\tilde{\mathbb{O}} \equiv diag(\vec{\tilde{A}}) \cdot \mathbb{D}_{zz} + diag(\vec{\tilde{B}}) \cdot \mathbb{D}_z + diag(\vec{\tilde{C}}) \cdot \mathbb{I}, \tag{3.8}$$

where $diag(\vec{\tilde{X}})$ is a diagonal matrix with the entries of the vector $\vec{\tilde{X}}$ on the diagonal and the coefficient vectors are defined as

$$\begin{cases} \tilde{X}_1 \equiv X_b \\ \tilde{X}_i \equiv X_i, \quad i = 2 \dots N-1 \\ \tilde{X}_N \equiv X_t \end{cases} \qquad X \in \{A, B, C, R\} \tag{3.9}$$

and we define $A_t = A_b \equiv 0$, since the boundary conditions must be first order.

Note that now all the information about the boundary value problem, **including boundary conditions** is encoded in the set of coefficient vectors $\vec{\tilde{X}}$.

Once the BVP operator (3.8) is constructed, the linear problem (3.6) can be solved by direct inversion:

$$\vec{f} = \tilde{\mathbb{O}}^{-1} \cdot \vec{\tilde{R}} \tag{3.10}$$

3.2 Assignment

Solve the equation

$$f''(z) - \pi f'(z)\cot(\pi z) = 0 \tag{3.11}$$

in the domain $z \in [0, 1]$ with the boundary conditions

$$f(0) = 1, \qquad f'(1) = 0. \tag{3.12}$$

Note that the equation is singular at the boundary $z = 1$.
Solution

$$f(z) = \cos(\pi z) \tag{3.13}$$

4 Nonlinear equations

In the previous section we considered the equations with coefficients depending on the coordinate z. It is straightforward to generalize this treatment to the case when the coefficients are dependent on the function itself – the nonlinear differential equations. The essential step to be made is to set up the iterative procedure for the nonlinear equation, which relies on solving the linearized system at every step.

4.1 Iterative solution to nonlinear differential equation: Newton method

Consider the nonlinear equation

$$\mathbf{E}\left[\partial_z^2 F(z), \partial_z F(z), F(z), z\right] = G(z), \tag{4.1}$$

where $\mathbf{E}$ is an arbitrary function. Assume F_0 is an exact solution to this equation and F_n is a close enough approximation to it. More precisely

$$F_n = F_0 + f, \qquad ||f|| \ll 1, \tag{4.2}$$

with some chosen norm $||*||$. Applying $\mathbf{E}$ to both sides leads to

$$\mathbf{E}\left[F_n\right] = G + \frac{\delta \mathbf{E}\left[F_n\right]}{\delta \partial_z^2 F}\partial_z^2 f(z) + \frac{\delta \mathbf{E}\left[F_n\right]}{\delta \partial_z F}\partial_z f(z) + \frac{\delta \mathbf{E}\left[F_n\right]}{\delta F} f(z) + O(f^2). \tag{4.3}$$

It can be recast in familiar form

$$\mathbf{A}[F_n, z]\partial_z^2 f(z) + \mathbf{B}[F_n, z]\partial_z f(z) + \mathbf{C}[F_n, z] f(z) = \mathbf{R}[F_n, z], \tag{4.4}$$

with

$$\mathbf{R}[F_n, z] \equiv \mathbf{E}[F_n, z] - G(z). \tag{4.5}$$

Once the linearized equation (4.4) is solved, one obtains the next, better, approximation to the solution:

$$F_{n+1} = F_n - f \tag{4.6}$$

and the procedure is reiterated up to the point when the nonlinear equation is satisfied to the desired accuracy δ, i.e.

$$||\mathbf{R}[F_n, z]|| = ||\mathbf{E}[F_n, z] - G(z)|| \ll \delta \tag{4.7}$$

This iterative method is known as **Newton method for iterative solution** of the nonlinear equation.

4.2 Variable coefficients

The only difference between the linearized equation (4.4) and the linear equation (3.1) is the fact that the coefficients are now dependent not only on the coordinate z, but also on the values of the background function F_n and its derivatives on the grid. Therefore, the evaluation of the coefficient vectors (3.9) breaks into two steps.

Firstly, given the n-th approximation $F_n(z_i)$ on the grid one uses a chosen method to obtain the values of $\partial_z F_n(z_i)$ and $\partial_z^2 F_n(z_i)$. For instance by applying the differentiation matrices (2.15).

Then one substitutes the vectors $\overrightarrow{F}, \overrightarrow{\partial_z F}, \overrightarrow{\partial_z^2 F}$ and the grid coordinates into the functions $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{R}$ in order to obtain the vectors of coefficients (3.4):

$$\mathbf{X}[F, z] \to \vec{X}, \qquad X_i \equiv \mathbf{X}[(\partial_z^2 F)_i, (\partial_z F)_i, F_i, z_i], \qquad X = \{A, B, C, R\} \tag{4.8}$$

Note that again, in order to construct the BVP operator, the boundary conditions should be substituted at the endpoints as in (3.9). The nonlinear boundary conditions can be linearized following exactly the same procedure as for the main equation. For every step in the iteration procedure, the BVP operator is constructed as

$$\tilde{\mathbb{O}}[F_n] \equiv diag(\vec{\tilde{A}}[F_n]) \cdot \mathbb{D}_{zz} + diag(\vec{\tilde{B}}[F_n]) \cdot \mathbb{D}_z + diag(\vec{\tilde{C}}[F_n]) \cdot \mathbb{I}. \tag{4.9}$$

and the subsequent approximation is obtained as

$$\vec{F}_{n+1} = \vec{F}_n - \tilde{\mathbb{O}}[F_n]^{-1} \cdot \vec{\tilde{R}}[F_n] \tag{4.10}$$

In the Newton method, the vectors of coefficients are recalculated at every iteration step. The *pseudo-Newton methods* exist, which take advantage of the fact that F_{n+1} is generically close to F_n and allow to use some of the coefficients evaluated at the previous steps.

4.3 Assignment

Solve the nonlinear equation

$$F''(z) - F(z)^2 = 2 + z^4 \tag{4.11}$$

in the domain $z \in [0, 1]$ with the boundary conditions

$$F(0) = 0, \qquad F(1) = 1. \tag{4.12}$$

Solution

$$F(z) = z^2 \tag{4.13}$$

5 System of equations

In the previous sections we were dealing with a single equation on a single function. How are the outlined procedures modified in case of a system of the coupled differential equations on several functions?

Consider the linear system of K equations on the functions $f^k(z)$, $k = 1 \dots K$. It can be represented as a K-valued vector differential equation

$$\mathcal{A} \cdot \begin{pmatrix} \partial_z^2 f^1(z) \\ \vdots \\ \partial_z^2 f^K(z) \end{pmatrix} + \mathcal{B} \cdot \begin{pmatrix} \partial_z f^1(z) \\ \vdots \\ \partial_z f^K(z) \end{pmatrix} + \mathcal{C} \cdot \begin{pmatrix} f^1(z) \\ \vdots \\ f^K(z) \end{pmatrix} = \begin{pmatrix} R^1(z) \\ \vdots \\ R^K(z) \end{pmatrix}, \tag{5.1}$$

where $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are $K \times K$-matrices of (coordinate dependent) coefficients and R^k is a K-vector of right hand sides.

The boundary conditions consist of K equations on every boundary, characterized similarly to (3.2) by the matrices of constant coefficients $\mathcal{B}_b, \mathcal{C}_b$ and $\mathcal{B}_t, \mathcal{C}_t$.

5.1 Flattened vectors

When we discretize this system on a lattice with N nodes, every function f^k (and its derivatives) turns to N-vector. Therefore $(f^1(z), \dots, f^K(z))^T$ turns into K-vector of N-vectors. This is not a very handy object. Instead we will arrange the values of all the functions on the grid as a single $K \cdot N$ **flattened** vector.

$$\begin{pmatrix} (f^1(z_1), \dots, f^1(z_N))^T \\ \vdots \\ \big(f^K(z_1), \dots, f^K(z_N)\big)^T \end{pmatrix} \xrightarrow{flatten} \begin{pmatrix} f^1(z_1) \\ \vdots \\ f^1(z_N) \\ f^2(z_1) \\ \vdots \\ f^K(z_N) \end{pmatrix} \tag{5.2}$$

5.2 Enlarged differential matrix

Given this data structure for the values of the functions, we wish to have similar objects for the derivatives. This requires defining the enlarged $K \cdot N \times K \cdot N$ differentiation matrices. Since the values of one function do not affect the derivatives of the other function, it is clear that the shape of the enlarged differentiation matrices will be block diagonal. For instance

$$\begin{pmatrix} \partial_z f^1(z_1) \\ \vdots \\ \partial_z f^1(z_N) \\ \partial_z f^2(z_1) \\ \vdots \\ \partial_z f^K(z_N) \end{pmatrix} = \begin{pmatrix} \mathbb{D}_z & 0 & \cdots & 0 \\ 0 & \mathbb{D}_z & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbb{D}_z \end{pmatrix} \cdot \begin{pmatrix} f^1(z_1) \\ \vdots \\ f^1(z_N) \\ f^2(z_1) \\ \vdots \\ f^K(z_N) \end{pmatrix}, \tag{5.3}$$

where $\mathbb{D}_z$ is the familiar $(N \times N)$ differentiation matrix on the grid (2.15). In a more concise notation, the enlarged differentiation matrix $\overline{\mathbb{D}}_z$ can be defined via the **Kronecker product**

$$\overline{\mathbb{D}}_z \equiv \mathbb{I}_{K\times K} \otimes \mathbb{D}_z \tag{5.4}$$

Here and in what follows we will use bar in order to distinguish the objects in the space of flattened vectors.

Note that Kronecker product is not commutative, there is a useful "rule of thumb" to remember the order of terms. It coincides with the order of indices in the "vector of vectors" structure in (5.2) before flattening: in order to access the value $f^k(z_i)$ one has to first take the k-th line and then the i-th value in this line.

5.3 Coefficients

The situation with coefficient $(K \times K)$-matrices $\mathcal{A}, \dots$ is a bit more convoluted. Upon discretization on the grid every entry of the matrix turns into an N-vector of corresponding values at the lattice nodes. These vectors should multiply the rows of the corresponding differentiation matrices as in (3.8). Therefore we similarly turn them into the diagonal $(N\times N)$-matrices. Eventually, the coefficients in the equations get arranged in the $(K \times K)$-block matrices with diagonal $(N \times N)$ blocks. These block matrices will further multiply the enlarged differentiation matrices in order to provide the $(K \cdot N \times K \cdot N)$ BVP operator.

Consider as example a system of 2 equations:

$$\begin{array}{ll} IE_1: & \begin{cases} A_{11}\partial_z^2 f_1 + B_{11}\partial_z f_1 + B_{12}\partial_z f_2 = R_1 \\ A_{22}\partial_z^2 f_2 + B_{21}\partial_z f_1 = R_2 \end{cases} \\ IE_2: & \end{array} \tag{5.5}$$

It can be recast in the form (5.1) with the coefficient matrices

$$\mathcal{A} = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}, \qquad \mathcal{B} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & 0 \end{pmatrix}, \qquad \mathcal{C} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{5.6}$$

Upon discretization, using the techniques developed in Sec.3 we can represent the first equation in (5.5) as an N-vector equation

$$diag(\overrightarrow{A_{11}}) \cdot \mathbb{D}_{zz} \cdot \vec{f}_1 + diag(\overrightarrow{B_{11}}) \cdot \mathbb{D}_z \cdot \vec{f}_1 + diag(\overrightarrow{B_{12}}) \cdot \mathbb{D}_z \cdot \vec{f}_2 = \overrightarrow{R_1} \tag{5.7}$$

Using the notation of the enlarged matrices we work with the whole system at once. The coefficient matrices turn into the block matrices

$$\overline{\mathcal{A}} = \begin{pmatrix} diag(\overrightarrow{A_{11}}) & 0 \\ 0 & diag(\overrightarrow{A_{22}}) \end{pmatrix}, \quad \overline{\mathcal{B}} = \begin{pmatrix} diag(\overrightarrow{B_{11}}) & diag(\overrightarrow{B_{12}}) \\ diag(\overrightarrow{B_{21}}) & 0 \end{pmatrix}, \quad \overline{\mathcal{C}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{5.8}$$

And the differentiation matrices are

$$\overline{\mathbb{D}_z^2} = \begin{pmatrix} \mathbb{D}_z^2 & 0 \\ 0 & \mathbb{D}_z^2 \end{pmatrix}, \qquad \overline{\mathbb{D}_z} = \begin{pmatrix} \mathbb{D}_z & 0 \\ 0 & \mathbb{D}_z \end{pmatrix}, \qquad \bar{\mathbb{I}} = \begin{pmatrix} \mathbb{I} & 0 \\ 0 & \mathbb{I} \end{pmatrix}. \tag{5.9}$$

The full system of equations takes the form

$$\overline{IE}: \qquad \left(\overline{\mathcal{A}}\cdot\overline{\mathbb{D}_z^2}+\overline{\mathcal{B}}\cdot\overline{\mathbb{D}_z}+\overline{\mathcal{C}}\cdot\overline{\mathbb{I}}\right)\cdot\overline{f}=\overline{R}, \tag{5.10}$$

where $\overline{f}=flatten[(\vec{f}_1,\vec{f}_2)^T]$ is a flattened vector (5.2) and $\overline{R}=flatten[(\overrightarrow{R_1},\overrightarrow{R_2})^T]$. By straightforward multiplication of block matrices one can check that the first N lines of (5.10) do indeed coincide with (5.7). The advantage of the flattened notation is that the whole system is represented as a single matrix equation (5.10).

5.4 Boundary conditions

The question about implementation of the boundary conditions becomes more subtle as well. The system of K differential equations on an interval requires $2K$ boundary conditions: one per function on two boundaries. In complete analogy with the treatment of Sec.3, in order to implement the boundary conditions as in (3.9) one has to substitute the first and last elements in the coefficient N-vectors $\vec{X}_{\alpha\beta}$ in (5.8) with the corresponding values of $(X_b)_{\alpha\beta},(X_t)_{\alpha\beta}$. This would produce the effective coefficient vectors $\vec{\tilde{X}}_{\alpha\beta}$, which can be used to construct the BVP operator and right hand side:

$$\overline{BVP}: \qquad \left(\overline{\tilde{\mathcal{A}}}\cdot\overline{\mathbb{D}_z^2}+\overline{\tilde{\mathcal{B}}}\cdot\overline{\mathbb{D}_z}+\overline{\tilde{\mathcal{C}}}\cdot\overline{\mathbb{I}}\right)\cdot\overline{f}=\overline{\tilde{R}}, \tag{5.11}$$

$$\overline{\tilde{\mathcal{X}}}=\begin{pmatrix} diag(\vec{\tilde{X}}_{11}) & \dots & diag(\vec{\tilde{X}}_{1K}) \\ \vdots & \ddots & \vdots \\ diag(\vec{\tilde{X}}_{K1}) & \dots & diag(\vec{\tilde{X}}_{KK}) \end{pmatrix}, \qquad \mathcal{X}\in\{\mathcal{A},\mathcal{B},\mathcal{C}\} \tag{5.12}$$

It is instructive to figure out what does this operation mean for the flattened vector of equations (5.10). Since this flattened vector has exactly the same structure as $\overline{f}$ in (5.2), we can *unflatten* it and represent as a K-vector of N-vectors of equations

$$\overline{IE}\equiv\begin{pmatrix} IE^1(z_1) \\ \vdots \\ IE^1(z_N) \\ IE^2(z_1) \\ \vdots \\ IE^K(z_N) \end{pmatrix} \xrightarrow{\text{unflatten}} \begin{pmatrix} (IE^1(z_1),\dots,IE^1(z_N))^T \\ \vdots \\ \big(IE^K(z_1),\dots,IE^K(z_N)\big)^T \end{pmatrix} \tag{5.13}$$

Then implementing the boundary conditions means that every entry $IE^\alpha(z_1)$ gets substituted with BC_b^α and $IE^\alpha(z_N)$ with BC_t^α. It is most easily achieved by working

on the *second level* of the "vector of vectors" structure and flattening it in the end

$$\overline{BVP} \equiv \begin{pmatrix} BC_b^1 \\ IE^1(z_2) \\ \vdots \\ BC_t^1 \\ BC_b^2 \\ IE^2(z_2) \\ \vdots \\ BC_t^K \end{pmatrix} = \text{flatten} \begin{pmatrix} (BC_b^1, IE^1(z_2), \ldots, BC_t^1)^T \\ \vdots \\ \left(BC_b^K, IE^K(z_2), \ldots, BC_t^K\right)^T \end{pmatrix} \tag{5.14}$$

6 Partial Differential Equations

In many regards the implementation of the partial differential equations is similar to the case of a system of equations discussed above. The obvious additional complication is the variety of the distinct differentiation operators and the structure of the corresponding differentiation matrices. We will consider the case of 2-dimensional PDE, generalization to the higher dimensions is straightforward. In 2-dimensions (denote them x and y) there are 6 distinct differentiation operators, including identity [1]

$$\partial_p \in \{\partial_{xx}, \partial_{yy}, \partial_{xy}, \partial_x, \partial_y, 1\}, \tag{6.1}$$

Therefore a 2D partial differential equation can be characterized by a set of 6 coefficients $X^p(x, y)$ (which replace A, B and C used in 1D (3.1)) and a right hand side:

$$PDE: \qquad \sum_p^6 X^p(x, y)\, \partial_p f(x, y) = R(x, y) \tag{6.2}$$

6.1 Flattened vectors

Now we are dealing with a 2-dimensional grid:

$$x \to x_i, \qquad i = 1 \ldots N \tag{6.3}$$

$$y \to y_j, \qquad j = 1 \ldots M \tag{6.4}$$

When discretized on this grid, the function becomes a level 2 array, which we can be expressed as a $(N \times M)$ matrix

$$f(x, y) \to \vec{\vec{f}}, \qquad \vec{\vec{f}} \equiv \begin{pmatrix} f(x_1, y_1) & \ldots & f(x_1, y_M) \\ \vdots & & \vdots \\ f(x_N, y_1) & \ldots & f(x_N, y_M) \end{pmatrix}, \qquad f_{ij} \equiv f(x_i, y_j) \tag{6.5}$$

[1] In case of a single dimension, considered above, there were only 3 of them, in 3-dimensional equation there are 10, and so on.

As in Sec.5 we have to turn this data to a vector first, therefore we introduce the flattened $N \cdot M$-vector

$$\overline{f} \equiv \text{flatten}(\vec{\vec{f}}) = \begin{pmatrix} f(x_1, y_1) \\ \vdots \\ f(x_1, y_M) \\ f(x_2, y_1) \\ \vdots \\ f(x_N, y_M) \end{pmatrix} \tag{6.6}$$

6.2 2D differentiation matrices

Similarly to (5.3) we have to introduce the enlarged differentiation matrices, suitable for this data structure. It is easy to understand how the y-derivative should look like. Since it does not mix the functions at different x-coordinates, it acts as a block diagonal matrix on a flattened vector

$$\begin{pmatrix} \partial_y f(x_1, y_1) \\ \vdots \\ \partial_y f(x_1, y_M) \\ \partial_y f(x_2, y_1) \\ \vdots \\ \partial_y f(x_N, y_M) \end{pmatrix} = \begin{pmatrix} \mathbb{D}_y & 0 & \cdots & 0 \\ 0 & \mathbb{D}_y & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbb{D}_y \end{pmatrix} \cdot \begin{pmatrix} f(x_1, y_1) \\ \vdots \\ f(x_1, y_M) \\ f(x_2, y_1) \\ \vdots \\ f(x_N, y_M) \end{pmatrix}. \tag{6.7}$$

Therefore, in complete analogy with (5.3), we can define it as a Kronecker product

$$\overline{\mathbb{D}_y} = \mathbb{I}_{N\times N} \otimes \mathbb{D}_y \tag{6.8}$$

The x-derivative might seem less trivial. For instance, the x-derivatives at point x_i, $\overrightarrow{(\partial_x f)}_i \equiv \{(\partial_x f)_{i1}, \ldots, (\partial_x f)_{iM}\}$, are obtained as a linear combination of M-vectors $\vec{f}_{i-1}$ and $\vec{f}_{i+1}$. Therefore the enlarged differentiation matrix should act on vectors as (compare to (2.15))

$$\begin{pmatrix} \partial_x f(x_1, y_1) \\ \vdots \\ \partial_x f(x_1, y_M) \\ \partial_x f(x_2, y_1) \\ \vdots \\ \partial_x f(x_N, y_M) \end{pmatrix} = \frac{1}{2\Delta x} \begin{pmatrix} -3\mathbb{I} & 4\mathbb{I} & -\mathbb{I} & \ldots & 0 \\ -\mathbb{I} & 0 & \mathbb{I} & \ldots & 0 \\ 0 & -\mathbb{I} & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & \mathbb{I} \end{pmatrix} \cdot \begin{pmatrix} f(x_1, y_1) \\ \vdots \\ f(x_1, y_M) \\ f(x_2, y_1) \\ \vdots \\ f(x_N, y_M) \end{pmatrix}, \tag{6.9}$$

Where $\mathbb{I}$ is a $M \times M$ identity matrix. Conveniently, this differentiation matrix can also be represented as a Kronecker product:

$$\overline{\mathbb{D}_x} = \mathbb{D}_x \otimes \mathbb{I}_{M\times M} \tag{6.10}$$

Similarly, all the other operators in (6.1) are represented by the matrices obtained as Kronecker products:

$$\overline{\mathbb{D}_{xx}} = \mathbb{D}_{xx} \otimes \mathbb{I}_{M\times M}, \qquad \overline{\mathbb{D}_{yy}} = \mathbb{I}_{N\times N} \otimes \mathbb{D}_{yy}, \qquad \overline{\mathbb{D}_{xy}} = \mathbb{D}_{x} \otimes \mathbb{D}_{y}, \tag{6.11}$$

$$\overline{\mathbb{D}_{x}} = \mathbb{D}_{x} \otimes \mathbb{I}_{M\times M}, \qquad \overline{\mathbb{D}_{y}} = \mathbb{I}_{N\times N} \otimes \mathbb{D}_{y}, \qquad \bar{\mathbb{I}} = \mathbb{I}_{N\times N} \otimes \mathbb{I}_{M\times M}. \tag{6.12}$$

The "rule of thumb" is the same as in (5.4). The first term in the product is an operator acting on the first index in the matrix form $\vec{\vec{f}}$ (6.6), which corresponds to x-coordinate and the second – operator acting on the second index, y. Analogously one can construct differentiation matrices in 3 dimensions. Theses will be the Kronecker products of 3 terms, corresponding to the operators acting on the 3 different coordinates.

6.3 Boundary conditions

Upon the discretization on the 2D grid the coefficients $X^p(x,y)$ turn to the matrices as in (6.5). Similarly to Sec.3 we implement the boundary conditions by substituting the corresponding entries in the matrices of coefficients.

In case of 2-dimensional problem one has 4 boundary conditions: top, bottom, left and right, characterized by the equations with coefficients $X^p_t(x), X^p_b(x), X^p_l(y)$ and $X^p_r(y)$ $(p = 1\dots 6)$, respectively. The corresponding boundaries on the grid are located at $y = y_M$ (top), $y = y_1$ (bottom), $x = x_1$ (left) and $x = x_N$ (right). The modified array of coefficients, which includes boundary conditions, is most easily constructed in the "matrix" notation:

$$\vec{\vec{X}} \to \vec{\tilde{\vec{X}}} \tag{6.13}$$

$$\vec{\vec{X}} \equiv \begin{pmatrix} X(x_1,y_1) & X(x_1,y_2) & \dots & X(x_1,y_{M-1}) & X(x_1,y_M) \\ X(x_2,y_1) & X(x_2,y_2) & \dots & X(x_2,y_{M-1}) & X(x_2,y_M) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ X(x_{N-1},y_1) & X(x_{N-1},y_2) & \dots & X(x_{N-1},y_{M-1}) & X(x_{N-1},y_M) \\ X(x_N,y_1) & X(x_N,y_2) & \dots & X(x_N,y_{M-1}) & X(x_N,y_M) \end{pmatrix} \tag{6.14}$$

$$\vec{\tilde{\vec{X}}} \equiv \begin{pmatrix} X_l(y_1) & X_l(y_2) & \dots & X_l(y_{M-1}) & X_l(y_M) \\ X_b(x_2) & X(x_2,y_2) & \dots & X(x_2,y_{M-1}) & X_t(x_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ X_b(x_{N-1}) & X(x_{N-1},y_2) & \dots & X(x_{N-1},y_{M-1}) & X_t(x_{N-1}) \\ X_r(y_1) & X_r(y_2) & \dots & X_r(y_{M-1}) & X_r(y_M) \end{pmatrix}, \tag{6.15}$$

As a result, the first and last rows and the first and last columns of the coefficient matrix $\vec{\vec{X}}$ are substituted by the corresponding boundary conditions in $\vec{\tilde{\vec{X}}}$. Same operation is performed with the right hand side terms $\vec{\vec{R}}$. It should be noted here

that consistency of the boundary conditions requires in the corners:

$$X_l(y_1) = X_b(x_1), \quad X_r(y_1) = X_b(x_N), \quad X_r(y_M) = X_t(x_N), \quad X_l(y_M) = X_t(x_1) \tag{6.16}$$

6.4 BVP operator

In the end of the day we need to construct the BVP operator. In complete analogy with Sec.3 this is done by row-wise multiplication of the differentiation matrices with the coefficients (3.8). Some care should be taken to match the coefficients at given grid point with the derivatives in this point, but this is accounted for by the way we construct the differentiation matrices (6.7), (6.9), (6.11), which return the flattened vectors of the derivatives, compatible with the flattened vectors of coefficients

$$\overline{\tilde{X}} \equiv \text{flatten}\left(\vec{\vec{X}}\right). \tag{6.17}$$

Therefore the BVP operator for the partial differential equation (6.2) with corresponding boundary conditions is computed as

$$\tilde{\mathbb{O}} = \sum_p^6 diag\left(\overline{\tilde{X}^p}\right) \cdot \overline{\mathbb{D}}_p \tag{6.18}$$

And the boundary value problem can be solved by a direct inversion

$$\overline{f} = \tilde{\mathbb{O}}^{-1} \cdot \overline{\tilde{R}} \tag{6.19}$$

6.5 Assignment

Solve the partial differential equation

$$\partial_x^2 F(x,y) + \partial_y^2 F(x,y) + \pi^2 F(x,y) = 0 \tag{6.20}$$

in the domain $(x,y) \in [0,1] \times [0,1]$ with the boundary conditions

$$F(x,0) = \sin(\pi x), \quad \partial_y F(x,1) = 0, \quad F(0,y) = 0, \quad F(1,y) = 0. \tag{6.21}$$

Solution

$$F(x,y) = \sin(\pi x) \tag{6.22}$$

7 Periodic boundary conditions

So far we have only addressed the boundary conditions which can be expressed as local equations on the functions and their derivatives at the edges of the calculation domain. The other important type of the boundary conditions are the periodic ones. The essence of the periodic boundaries is the **identification** of the endpoints of the

calculation interval. Once such an identification is performed, no extra information, like special equations for the boundaries, is needed.

Take the homogeneous calculation grid on the interval $z \in [z_b, z_t)$ with identified endpoints. Similarly to (2.5) we can define

$$z_i = z_b + \frac{z_t - z_b}{N'}(i-1), \qquad i = 1 \dots N'. \tag{7.1}$$

Note that now the boundary point z_t **is not included** in the grid. Reason for this is due to the fact that we identify the function at the endpoints z_t and z_b

$$f(z_t) \equiv f(z_b), \tag{7.2}$$

So there is no reason to solve for the value $f(z_b)$ twice, while looking for a solution. Note also that the new definition (7.1) leads to N' points in the calculation grid (excluding z_t).

7.1 Differentiation matrices

The identification $f(z_t)$ and $f(z_b)$ requires a new type of the differentiation matrices. Consider the example with nearest neighbour approximation used in (2.15): $(\partial_z f)_i = (f_{i+1} - f_{i-1})/2\Delta z$. Inside the calculation domain no modifications are needed, since for i in $2 \dots N' - 1$ the neighbouring points f_{i-1} and f_{i+1} exist. What will happen near the boundaries? When $i = 1$ we would have $(\partial_z f)_1 = (f_2 - f_{1-1})/2\Delta z$. There is no point z_0 in the domain, so $f(z_0)$ doesn't make sense. Instead, keeping in mind that we identified f_1 with $f_{N'+1}$, we substitute f_{1-1} with $f_{N'}$ and get

$$(\partial_z f)_1 = \frac{(f_2 - f_{N'})}{2\Delta z} \qquad (\partial_z f)_{N'} = \frac{(f_1 - f_{N'-1})}{2\Delta z}. \tag{7.3}$$

Same "cyclic rule" is applied for any differences with any number of neighbours. The differentiation matrix corresponding to this rule looks simple

$$\mathbb{D}_z = \frac{1}{2\Delta z} \begin{pmatrix} 0 & 1 & 0 & \dots & 0 & 0 & -1 \\ -1 & 0 & 1 & \dots & 0 & 0 & 0 \\ 0 & -1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & -1 & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & -1 & 0 \end{pmatrix} \tag{7.4}$$

Note the values in the upper right and lower left corners.

Conveniently, once the periodic differentiation matrices are found, the implementation of the periodic boundary conditions is completed. Since the boundaries of the periodic domain are not in any way different from any internal points, one has

to solve the same equations of motion there. Therefore there is no need to substitute endpoints of the coefficient vectors as it was done in (3.9). In the case of multiple dimensions, the appropriate matrices are chosen for each coordinate and then the enlarged matrices are again created as a Kronecker product.

7.2 Assignment

Solve the partial differential equation

$$\partial_x^2 F(x,y) + \partial_y^2 F(x,y) + 4\pi^2 F(x,y) = 0 \tag{7.5}$$

in the domain $(x,y) \in [0,1] \times [0,1]$ with the boundary conditions

$$F(x,0) = \sin(2\pi x), \quad \partial_y F(x,1) = 0, \quad \text{periodic in } x. \tag{7.6}$$

Solution

$$F(x,y) = \sin(2\pi x) \tag{7.7}$$

8 Relaxation and preconditioning

All the preceding sections explained how to reduce the different kinds of the boundary value problems on the discrete grid to the linear matrix equations (2.24), (3.6), (5.11), (6.19), or the iterative series of them in case of nonlinear BVP (4.10). The most obvious way to solve such a system is to directly invert the linear operator. Unfortunately in practice this can be extremely demanding, since the matrices to be inverted can become large for large multidimensional grids and, for the higher order discretization schemes, dense. In this case the iterative relaxation approach can be useful.

8.1 Relaxation

Consider the linear system

$$\mathbb{O} \cdot \vec{f} = \vec{G}. \tag{8.1}$$

The essence of the relaxation method is to substitute (8.1) with the time evolution equation

$$\partial_t \vec{f} = \mathbb{O} \cdot \vec{f} - \vec{G}. \tag{8.2}$$

Clearly, the time evolution stops when (8.1) is satisfied. It's instructive to study in more detail though, how exactly the solution is approached.

Assume that f^0 is an exact t-independent solution $\mathbb{O} \cdot \vec{f^0} \equiv \vec{G}, \partial_t \vec{f^0} = 0$ at the later stage of the evolution (8.2) we can represent the time-dependent solution as $\vec{f}(t) = \vec{f^0} + \delta\vec{f}(t)$, where time-dependent residue $\delta\vec{f}(t)$ is small. Plugging this back into (8.2) we find

$$\partial_t \, \delta\vec{f} = \mathbb{O} \cdot \delta\vec{f}. \tag{8.3}$$

Consider the eigenfunctions g^k of the operator $\mathbb{O}$ with the eigenvalues λ^k:

$$\mathbb{O} \cdot g^k = \lambda^k g^k. \tag{8.4}$$

One can expand the residual function δf in a basis of these eigenfunctions

$$\delta f(t) = \sum_k c^k(t) g^k \tag{8.5}$$

Plugging this in (8.3) we get simply

$$\partial_t c^k(t) = \lambda^k c^k \qquad \Rightarrow \qquad c^k(t) \sim \exp(\lambda^k t). \tag{8.6}$$

It is remarkable, that for an elliptic (i.e. Laplace) equation with a negatively definite operator with $\lambda^k < 0, \forall k$ the components of the residual function decay exponentially, therefore the relaxation it this case does actually converge to the static solution! Notice that the speed of convergence is set by the **lowest** eigenvalue $\delta f \sim \exp(\lambda_{\min})$, $\lambda_{\min} = -\mathrm{Min}[|\lambda^k|, \forall k]$

In practice one discretizes the time derivative in (8.2) and setups the iterative procedure:

$$f^{n+1} = f^n + \delta t\, \mathbb{O} \cdot f^n - \delta t \vec{G}. \tag{8.7}$$

The main advantage of the relaxation method becomes obvious here: one does not need to invert $\mathbb{O}$ matrix at any point of the calculation!

It can be shown [5] that in order for the iteration to be numerically stable the value of the time step δt should be limited by the **highest** eigenvalue $\lambda_{\max}$ of $\mathbb{O}$

$$\delta t \sim \frac{1}{\lambda_{\max}}, \tag{8.8}$$

Therefore the number of iterations needed to achieve the solution, set by the slowest mode is proportional to $\lambda_{\max}/\lambda_{\min}$. This is a serious drawback since for $N \times N$ matrix $\mathbb{O}$ this ratio is typically of order N^2. Therefore the trivial relaxation procedure (8.7) is extremely ineffective.

8.2 Preconditioning

The relaxation procedure would work much faster if one would be able to use the operator with $\lambda_{\max}/\lambda_{\min} \approx 1$. One way to improve the situation is to note that instead of the relaxation equation (8.2) one can equally well use

$$\partial_t \vec{f} = -\hat{\mathbb{O}}^{-1} \cdot \left(\mathbb{O} \cdot \vec{f} - \vec{G} \right), \tag{8.9}$$

where $\hat{\mathbb{O}}$ is an arbitrary linear operator. Indeed, similar to (8.2), the evolution (8.9) will only stop when the solution to (8.1) is achieved. The iterative procedure will now take the form

$$f^{n+1} = f^n - \delta t\, \hat{\mathbb{O}}^{-1} \cdot \mathbb{O} \cdot f^n + \delta t \hat{\mathbb{O}}^{-1} \cdot \vec{G}, \tag{8.10}$$

and the time-step will be limited now by the highest eigenvalue of the regulated operator $\hat{\mathbb{O}}^{-1} \cdot \mathbb{O}$. This is the essence of the **preconditioning** procedure and the matrix $\hat{\mathbb{O}}$ is a **preconditioner matrix**.

It is easy to figure out that the perfect preconditioner, which will set $\lambda_{\max}/\lambda_{\min} = 1$ is an operator itself: $\hat{\mathbb{O}} = \mathbb{O}$. In this case the time step can be chosen as large as $\delta t = 1$ and the iteration (8.10) degenerates to a direct solution of (8.1) in one step. This is again not extremely practical, since inversion of the operator $\mathbb{O}$ is exactly something that we'd like to avoid.

But it's clear now how to use preconditioning to one's advantage. The preconditioner should be chosen in such a way that the highest eigenvalue of the product $\hat{\mathbb{O}}^{-1} \cdot \mathbb{O}$ is as close to unity as possible, while simultaneously the matrix $\hat{\mathbb{O}}$ is easily invertable. In case when $\mathbb{O}$ is constructed using some high order finite difference approximation, the good choice is to use the BVP operator of the same system, but constructed using **low** order finite difference approximation, for instance the nearest neighbour. This is known as an Orszag preconditioning [5]. This preconditioner approximates well the highest eigenvalues of $\mathbb{O}$, leading to $\delta t \approx 1$, but is also easily invertable being a sparse matrix due to the nearest neighbour finite difference scheme. The practical advantage of the Orszag preconditioning is also in the fact that since the operator $\hat{\mathbb{O}}$ has the same coefficients as $\mathbb{O}$, the effects of the singular terms in the equations gets diminished in the combination $\hat{\mathbb{O}}^{-1} \cdot \mathbb{O}$. Importantly, the low accuracy of the derivative approximation in $\hat{\mathbb{O}}$ doesn't affect the accuracy of the final solution, which is dictated by $\mathbb{O}$.

Importantly, the general analysis of the convergence of the relaxation schemes [5] shows that in any case the time step should be less then 1. For completeness, referring the reader to [5] for the details, we should mention here that one should use

$$\tau \leq \frac{4}{7} \tag{8.11}$$

for a the stable iteration.

The practical recipe therefore is to use some high order derivative discretization scheme for the operator $\mathbb{O}$ in order to achieve high accuracy of the solution, but use the low order scheme for $\hat{\mathbb{O}}$, making it easily invertable. Then, just a few step of the iteration (8.10) are enough to get the solution with desired precision.

Another practical advantage of the preconditioned relaxation outlined above shows up when one deals with the nonlinear equations discussed in Sec.4. Reason is that the solution of a nonlinear equation requires iteration anyway and there is no point to solve the equation exactly at every step. Therefore the nonlinear iteration can be easily combined with the relaxation iteration. Form this point of view one can regard the relaxation with preconditioning as a special version of a pseudo-Newton method, where at every step one inverts a certain approximation to $\mathbb{O}$ instead of the operator itself.

8.3 Assignment

Solve the problem of from one of the previous Sections with order 6 approximation for the derivatives in the equation operator. Use the direct inversion, then trivial relaxation (8.7) and in the end the relaxation with Orszag preconditioning. Compare the efficiencies of the applied methods.

9 Pseudospectral method

As it has been pointed out in the previous Section, the accuracy of the final solution is governed by the accuracy of the derivatives approximation used in the operator $\mathbb{O}$ in the right hand side of (8.2). It is therefore important to develop a procedure to approximate the derivatives accurately.

9.1 Pseudospectral collocation

The **pseudospectral collocation** method resides on the fact that instead of representing the function via its values on the grid, one can represent it as a series in the basis functions on a particular interval, giving the required values on the grid points. Consider for instance the interval $\theta \in [0, 2\pi)$ with periodic boundary conditions. In these conditions the function can be represented as a Fourier series

$$f(\theta) = \sum_k \hat{v}_k e^{ik\theta} \tag{9.1}$$

Given the values of the function on a homogeneous grid $v_i = f(\theta_i)$ one can unambiguously evaluate the coefficients $\hat{v}_k$. Once the coefficients are known, the derivative of the series (9.1) can be evaluated exactly:

$$w_i \equiv \partial_\theta f(\theta_i) = \sum_k ik\hat{v}_k e^{ik\theta_i} \tag{9.2}$$

Given that in order to figure out the coefficients c_n the values of function on the whole grid are used, the differentiation matrices corresponding to the pseudospectral collocation method are dense. The method might be seen as N-th order finite difference derivative for N-point grid.

Similar expansion can be done for the interval with the boundaries $z \in [-1, 1]$. in this case the basis functions are Chebyshev polynomials $T_k(z)$ and the unknown function can be represented as $F(z) = \sum V_k T_k$. The differential matrices are similarly dense in this case.

It looks like the pseudospectral methods deliver better accuracy for the price of having very inconvenient form of the differentiation matrices. This can be tolerated if one uses the relaxation discussed in Sec.8, since in this case there is no need to invert the matrix. But the bigger advantage of pseudospectral approach is unveiled once the efficient Fourier transform technique is used instead of the differentiation matrix multiplication.

9.2 Pseudospectral collocation and Fourier transform

Indeed, on a circle the coefficients of the Fourier series (9.1) can be evaluated using the discrete Fourier transform. Given the values of the function on the homogeneous grid v_i, the Fourier coefficients are [4]:

$$\hat{v}_k = \delta\theta \sum_{j=1}^{N} e^{-ik\theta_j} v_j, k = -\frac{N}{2} + 1, \ldots, \frac{N}{2} \tag{9.3}$$

and the Fourier coefficients for the derivative are simply

$$\hat{w}_k = ik\hat{v}_k. \tag{9.4}$$

In order to obtain the values of the derivative on the grid points one has to make another, inverse Fourier transform

$$w_j = \frac{1}{2\pi} \sum_{k=-N/2+1}^{N/2} e^{ik\theta_j} \hat{w}_k, \qquad j = 1, \ldots, N. \tag{9.5}$$

The great advantage of the pseudospectral method is the fact that one can perform the operations (9.3) and (9.5) efficiently using Fast Fourier Transform (FFT) algorithm and avoid using the nasty differentiation matrices all together.

Similarly, one can use FFT in order to evaluate the derivative of the function on an interval $z \in [-1, 1]$ represented as a series of Chebyshev polynomials. The distinctive feature of the Chebyshev polynomials on an interval is the fact that they can be directly related to the the harmonic functions on a circle. Given one identifies $z \equiv \cos(\theta)$ the n-th order polynomial is represented as

$$T_n(z) = \cos(n\theta), \qquad z \equiv \cos(\theta) \tag{9.6}$$

Therefore one can setup a one-to-one correspondence between the values of a symmetric function on a homogeneous grid on a circle

$$v_i \equiv f(\theta) = \sum \hat{v}_k \cos(k\theta_i), \qquad \theta_i = 0, \delta\theta, 2\delta\theta, \ldots \tag{9.7}$$

and the values of a function on a "Chebishev grid" (see Fig.1) on an interval

$$V_i \equiv F(z) = \sum \hat{V}_k T_k(z_i), \qquad z_i = 1, \cos(\delta\theta), \cos(2\delta\theta), \ldots \tag{9.8}$$

, with the same set of coefficients $\hat{V}_k = \hat{v}_k$.

After this identification is done one can evaluate the derivatives of $f(\theta)$ with the FFT, and once it is done, relate it to the derivatives of $F(z)$:

$$\partial_z F(z) = \partial_z f(\theta) = \frac{\partial z}{\partial \theta} \partial_\theta f(\theta) = \frac{\partial_\theta f(\theta)}{\sqrt{1 - z^2}}. \tag{9.9}$$

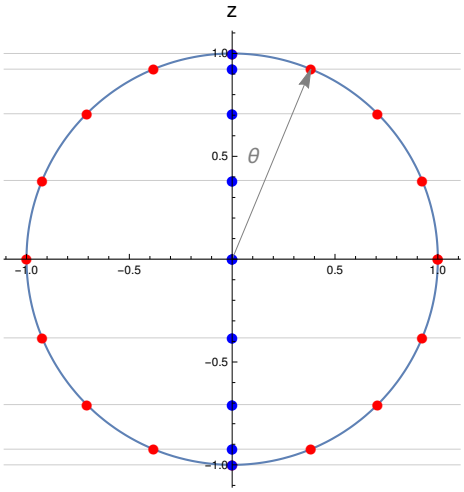

Figure 1. Relation between homogeneous grid on $\theta \in [0, 2\pi)$ and Chebyshev grid on $z \in [-1, 1]$

This formula will work for all the point on the interval excluding endpoint $z = 1, z = -1$, where the derivative is obtained using l'Hôpital's rule.

This is the final ingredient for a pseudospectral method. In the end of the day we see, that it is possible to setup the very efficient procedure using the relaxation technique, low order difference preconditioner and high order pseudospectral approximation to the BVP operator $\mathbb{O}$, implemented through the FFTs.

10 Conclusion and implementation

The outlined techniques one can reduce the nonlinear differential equation boundary value problem to the problem in linear algebra. In this form is it relatively straightforward to implement these procedures in one's favorite computing software. The relaxation procedure and the pseudospectral collocation technique require no more then the efficient sparse linear solver and fast Fourier transform. These routines can be find in any contemporary computation package or library including Mathematica, MATLAB, Python, FORTRAN etc. We do not discuss the implementation here leaving the choice to the reader. The methods discussed above, implemented in Wolfram Mathematica [7], were successfully applied to several numerical projects in applied AdS/CFT including 1-dimensional and 2-dimensional problems [8–14] and proved to be effective.

Acknowledgments

I appreciate contribution and enthusiastic support of the participants of the Numerical Study group: Aurelio Romero-Bermudez, Philippe Sabella-Garnier, Floris Balm

and Koenraad Schalm. I'm also grateful to Tomas Andrade in collaboration with whom most of the numerical projects were completed, where I handled the methods discussed here.

References

[1] G. T. Horowitz, J. E. Santos and D. Tong, *Optical Conductivity with Holographic Lattices*, *JHEP* **07** (2012) 168 [1204.0519].

[2] M. Rozali, D. Smyth, E. Sorkin and J. B. Stang, *Holographic Stripes*, *Phys. Rev. Lett.* **110** (2013), no. 20 201603 [1211.5600].

[3] A. Donos and J. P. Gauntlett, *The thermoelectric properties of inhomogeneous holographic lattices*, *JHEP* **01** (2015) 035 [1409.6875].

[4] L. N. Trefethen, *Spectral methods in MATLAB*, vol. 10. Siam, 2000.

[5] J. P. Boyd, *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.

[6] W. L. Briggs, V. E. Henson and S. F. McCormick, *A multigrid tutorial*. SIAM, 2000.

[7] Wolfram Research, Inc., *Mathematica, Version 10.2*. Champaign, Illinois, 2015.

[8] T. Andrade, A. Krikun, K. Schalm and J. Zaanen, *Doping the holographic Mott insulator*, 1710.XXXXX.

[9] A. Krikun, *Holographic discommensurations*, 1710.XXXXX.

[10] T. Andrade and A. Krikun, *Commensurate lock-in in holographic non-homogeneous lattices*, *JHEP* **03** (2017) 168 [1701.04625].

[11] T. Andrade and A. Krikun, *Commensurability effects in holographic homogeneous lattices*, *JHEP* **05** (2016) 039 [1512.02465].

[12] A. Krikun, *Phases of holographic d-wave superconductor*, *JHEP* **10** (2015) 123 [1506.05379].

[13] T. Andrade, M. Baggioli, A. Krikun and N. Poovuttikul, *Pinning of longitudinal phonons in holographic helical crystals*, 1708.08306.

[14] A. Gorsky, S. B. Gudnason and A. Krikun, *Baryon and chiral symmetry breaking in holographic QCD*, *Phys. Rev.* **D91** (2015), no. 12 126008 [1503.04820].