MIGSAA Extended-Project Author's Declaration

Help and/or advice was received from the following people.

Supervisor(s) : Dr Ben Goddard
Dr John Pearson

Any Others (list any names):

Mildred Aduamoah

I affirm that this report is my sole and original work and that any people who gave assistance are listed above.		
Author Name (PRINT)	**Signed**	**Date**
Jonna Rader	J. Radee	30/07/2019

PDE-Constrained Optimization for Multiscale Particle Dynamics

with Industrial Applications

Jonna C. Roden

Supervision by Dr Ben Goddard and Dr John Pearson

MIGSAA

July 30, 2019

Abstract

Numerical solutions to PDE-constrained optimization problems are of increasing interest to academia and industry. In particular, there are many industrial processes that are best described by the integro-PDEs for multiscale particle dynamics. Solving PDE-constrained optimization problems involving integro-PDE constraints requires the development of new numerical methods. This report addresses the derivation of such a numerical method. First, the PDE for the particle dynamics is discussed. Then, the optimization problem is posed and optimality conditions for the solution of this problem are derived. Concepts, such as pseudospectral methods and the multiple shooting method, are introduced and applied to the numerical method. Finally, some numerical experiments are presented for one- and two-dimensional test problems. The findings of this report contribute to achieving the numerical optimization of industrial processes, such as beer brewing and nano filtration. The industrial partners are WEST brewery and ufraction8.

Acknowledgements

I would like to thank my two supervisors Dr Ben Goddard and Dr John Pearson for their time and support during this project. I really appreciate the detailed and simple explanations of many complex concepts and the help in writing and debugging the Matlab code. Furthermore, I would like to thank Mildred Aduamoah for her help and the work we did together on the numerical method.

Contents

1 Introduction

PDE-constrained optimization and multiscale particle dynamics are both fields of growing interest to academia and industry. Applying methods of PDE-constrained optimization to industrial processes is a highly relevant topic, for example in the oil and gas industry [1], the beer industry [2] and the wine industry [3]. There are two industrial partners affiliated with this project. WEST brewery is interested in optimizing the yeast sedimentation in the beer brewing process. The company ufraction8 works on cell separation and nano filtration devices, which separate particles based on their sizes. They are interested in optimizing this process.

Many processes, including these two examples, can be described as interacting particle systems, using Density Functional Theory (DFT). Further examples include processes in biology and nanotechnology [4], as well as in chemical engineering [5]. Therefore, developing a numerical framework for PDE-constrained optimization problems, where the PDE constraint describes particle dynamics, is highly relevant. This task is challenging, because the PDEs involved are non-local, nonlinear integro-PDEs. This makes the application of standard methods, such as finite element methods, hard. Pseudospectral methods can be used to tackle the numerical challenges, such as non-local boundary conditions and dense matrices in the discretized problem. In this report, steps towards developing this numerical method are taken by deriving a method for one- and two-dimensional test problems, which can be extended to the full problem, including the PDE describing the particle dynamics.

The report is structured as follows. In the next section, the PDE for the particle dynamics is derived from the Smoluchowski equation. In the third section, the PDE-constrained optimization framework is set up and first-order optimality conditions are derived for two different test problems, one diffusion type problem and one problem including an integral particle interaction term. In the fourth section, some exact solutions and the numerical methods are introduced, such as pseudospectral methods and the multiple shooting method. In the fifth section, some numerical experiments are presented for both introduced test problems in one and two dimensions. In the final section, some conclusions are drawn and opportunities for future work are pointed out.

2 Multiscale Particle Dynamics

The aim of this section is to model the dynamics of particles, which are suspended in a bath. The particles evolve in time and in some domain Ω in space. Instead of modelling the movement of a particular particle, a space coordinate $r \in \Omega$ is fixed. The probability of a particle being at the position r at time t is modelled as a distribution $\rho(r,t)$, the one-body particle density. The model equations are:

$$\begin{aligned} \partial_t \rho(r,t) &= \nabla \cdot \left(\big(\nabla + \nabla V_{ext} + \int_\Omega \rho(r') \nabla V_2(|r-r'|) dr' - \mathbf{w}\big)\rho(r,t) \right), \\ &:= \nabla \cdot \mathbf{j}, \end{aligned} \tag{1}$$

$$\begin{aligned} \mathbf{j} \cdot \mathbf{n} &= 0 \quad \text{on} \quad \partial\Omega, \\ \rho(0) &= \rho_0 \quad \text{at} \quad t = t_0, \end{aligned}$$

where $\mathbf{j} = \big(\nabla + \nabla V_{ext} + \int_\Omega \rho(r') \nabla V_2(|r-r'|) dr' - \mathbf{w}\big)\rho(r,t)$ and $\mathbf{n}$ denotes the outward normal to $\partial\Omega$, compare to [6]. The equation describes the particle dynamics, including a time derivative, a diffusion term, and an external potential V_{ext}, whose negative gradient is a body force such as gravity. Furthermore, there is a term describing the background flow field of the bath, $\mathbf{w}$, and a particle interaction term.

This particle interaction term describes the interactions of two particles at the position r and some other position $r' \in \Omega$, where $r' \neq r$. The forces between the particles at r and r' are represented by $-\nabla V_2(|r-r'|)$. This is multiplied by the particle density at r', $\rho(r')$, and the particle density at r, $\rho(r)$. Since each position $r' \in \Omega$ needs to be considered, the resulting expression is integrated over all r'.

The particle interaction in (1) is restricted to two-body interactions and can be extended to three- or more body interaction terms, which makes the model equations more accurate and complex. The inclusion of higher-order terms is dependent on the application. Furthermore, the forces between the particles included here only describe interactions such as attraction and repulsion. The model neglects hydrodynamic interactions, which are the effects of the particles moving in the bath. This movement causes a change in the flow field, which in turn influences the surrounding particles. This is a non-local phenomenon, which is more complex to model, however, significant to include in various applications.

2.1 Deriving Model Equations

The model equation (1) can be derived from the full N-body particle distribution. This has been done by M. Rex and H. Loewen [6], whose derivation is followed in this section. On the microscopic level, a probability distribution $P(r^N, t)$ of the total number N particles in the bath

at time t is considered, where $r^N = (r_1, r_2, ..., r_N)$, and $r_i \in \mathbf{R}^3$, $i = 1, 2, ..., N$. The dynamics of the probability distribution $P(r^N, t)$ can be described by the Smoluchowski equation, as presented in [6]:

$$\partial_t P(r^N, t) = \sum_{i,j}^{N} \nabla_i \cdot \left(D_{i,j}(r^N) \left(\nabla_j + \frac{\nabla_j U(r^N, t)}{k_B T} - \mathbf{w}(r_i) \right) P(r^N, t) \right), \tag{2}$$

where ∇_i refers to the operation with respect to the coordinate r_i. The term $D_{i,j}(r^N)$ is the diffusion tensor, which describes the hydrodynamic interactions of the particles at r_i and r_j, $U(r^N, t)$ is a potential, which could include an external potential and particle interactions, T is the temperature, and k_B is the Boltzmann constant. A background flow is defined by $\mathbf{w} \in \mathbf{R}^3$, describing the flow of the bath fluid. Note that this version of the equation is more general than the representation in the paper, due to the extra term $\mathbf{w}$. The aim is to derive a three dimensional approximation $\rho^{(1)}(r_1, t)$ to (2). The following n-body densities are defined as in [6]:

$$\begin{aligned} \rho^{(1)}(r_1, t) &= N \int_\Omega dr_2 .... dr_N P(r^N, t) := \rho(r, t), \\ &\vdots \\ \rho^{(n)}(r_n, t) &= \frac{N!}{(N-i)!} \int_\Omega dr_{n+1} .... dr_N P(r^N, t), \end{aligned}$$

where $\Omega = \mathbf{R}^{3N}$. The n-body densities are derived by integrating the full N particle probability distribution over $r_{n+1}, ...r_N$, and multiplying it by a prefactor. This definition is chosen to suit the computations, which are detailed below. In order to derive a three-dimensional approximation in terms of the one-body density $\rho(r, t)$, initially a simplification of (2) is considered and then extended in later sections to derive the approximation for the full system.

2.1.1 The Diffusion Equation

Considering $D_{i,j} = \delta_{i,j}$, $U = 0$ and $\mathbf{w} = 0$ in (2), the diffusion equation is recovered:

$$\begin{aligned} \partial_t P(r^N, t) &= \sum_{i}^{N} \nabla_i \cdot \left(\nabla_i P(r^N, t) \right) = \sum_{i}^{N} \Delta_i P(r^N, t) \\ &= \Delta_{r^N} P(r^N, t), \end{aligned}$$

where $\sum_i^N \Delta_i := \Delta_{r^N}$.
In order to derive the one-body density $\rho(r, t)$ for the diffusion equation, the equation is multiplied by N and integrated over all other positions $r_2, ..., r_N$:

$$\int_\Omega N \partial_t P(r^N, t) dr_2 ... dr_N = \int_\Omega N \sum_{i}^{N} \nabla_i \cdot \left(\nabla_i P(r^N, t) \right) dr_2 ... dr_N.$$

The integration is only dependent on space, not on time, so that the time derivative can be taken out of the integral. Furthermore, the sum on the right-hand side of the equation, as well as the integration, is finite. Therefore, Fubini's Theorem can be used to take the sum out of the integral. The equation is then:

$$N\partial_t \int_\Omega P(r^N,t)dr_2...dr_N = N\sum_i^N \int_\Omega \nabla_i \cdot \Big(\nabla_i P(r^N,t)\Big)dr_2...dr_N.$$

The Divergence Theorem can be applied to $i = 2,...,N$, while for $i = 1$ the integral remains unchanged, since the integration is independent of r_1. The equation is now:

$$N\partial_t \int_\Omega P(r^N,t)dr_2...dr_N = N\sum_{i=2}^N \int_{\partial\Omega} \frac{\partial P(r^N,t)}{\partial n}dr_2...dr_N + N\int_\Omega \nabla_1 \cdot \Big(\nabla_1 P(r^N,t)\Big)dr_2...dr_N,$$

where n is the outward normal. Now, the boundary condition for $P(r^N,t)$ can be employed. Since P is a probability distribution over a finite number of positions r_i, on an infinite domain $\Omega = \mathbf{R}^{3N}$, the natural boundary condition is that P and its derivatives vanish on the boundary $\partial\Omega$, i.e. at infinity. Furthermore, considering the fact that ∇_1 is constant with respect to the integration variables, it can be taken out of the integral and the following result is found, where the definition $\rho(r,t) = N\int_\Omega P(r^N,t)dr_2...dr_N$ is used:

$$\begin{aligned}\partial_t\rho(r,t) &= \nabla_1 \cdot \Big(\nabla_1 N\int_\Omega P(r^N,t)\Big)dr_2...dr_N,\\ &= \nabla_1 \cdot (\nabla_1\rho(r,t)).\end{aligned}$$

This is a one-body diffusion equation in $\mathbf{R}^3$, which is an approximation to the diffusion equation of the full N particle probability distribution $P(r^N,t)$. In a last step, the subscript of ∇_1 can be dropped, since the only position considered in this equation is r_1. The final equation is

$$\partial_t\rho(r,t) = \nabla\cdot(\nabla\rho(r,t)). \tag{3}$$

2.1.2 Adding Pairwise Interactions

After deriving the one-body equation for the diffusion term, a more complex version of (2) can be considered. Let $D_{i,i} = \delta_{i,j}$, as before, but define U as:

$$U = \sum_{m=1}^N V_{ext}(r_m,t) + \frac{1}{2}\sum_{m\neq n}^N V_2(|r_m - r_n|),$$

where V_{ext} is an external potential and V_2 is the potential due to forces between two particles. The PDE considered is again a simplified version of (2) and has the form:

$$\partial_t P(r^N,t) = \sum_i^N \nabla_i \cdot \Bigg(\Big(\nabla_i + \nabla_i\Big(\sum_{m=1}^N V_{ext}(r_m,t) + \frac{1}{2}\sum_{m\neq n}^N V_2(|r_m - r_n|)\Big)\Big)P(r^N,t)\Bigg), \tag{4}$$

where $k_BT = 1$ for simplicity. The diffusion term in the equation can be treated equivalently to the procedure in the previous section. The two new terms are treated similarly. The equation is multiplied by N and integrated over $r_2...r_N$. This gives:

$$
\begin{aligned}
&N\int_\Omega \partial_t P(r^N,t)dr_2...dr_N\\
=&N\int_\Omega \sum_i^N \nabla_i\cdot\Bigg(\Bigg(\nabla_i+\nabla_i\Bigg(\sum_{m=1}^N V_{ext}(r_m,t)+\frac{1}{2}\sum_{m\neq n}^N V_2(|r_m-r_n|)\Bigg)\Bigg)P(r^N,t)\Bigg)dr_2...dr_N,\\
=&N\int_\Omega \sum_i^N \nabla_i\cdot\nabla_i P(r^N,t)dr_2...dr_N+N\int_\Omega\sum_i^N\nabla_i\cdot\Bigg(P(r^N,t)\nabla_i\sum_{m=1}^N V_{ext}(r_m,t)\Bigg)dr_2...dr_N\\
+&N\int_\Omega \sum_i^N \nabla_i\cdot\Bigg(P(r^N,t)\nabla_i\frac{1}{2}\sum_{m\neq n}^N V_2(|r_m-r_n|)\Bigg)dr_2...dr_N\\
:=&I_1+I_2+I_3.
\end{aligned}
$$

The left-hand side satisfies $N\int_\Omega \partial_t P(r^N,t)dr_2...dr_N = \partial_t\rho(r,t)$ from the previous section. The integral I_1, by (3), satisfies:

$$I_1 = N\int_\Omega\sum_i^N\nabla_i\cdot\nabla_i P(r^N,t)dr_2...dr_N = \Delta\rho(r,t). \tag{5}$$

Next, the integral I_2 is considered:

$$I_2 = N\int_\Omega\sum_i^N\nabla_i\cdot\Bigg(P(r^N,t)\nabla_i\sum_{m=1}^N V_{ext}(r_m,t)\Bigg)dr_2...dr_N.$$

By the same argument as in the previous section, the integration and summation can be swapped. Furthermore, $\nabla_i\sum_{m=1}^N V_{ext}(r_m,t) = \nabla_i V_{ext}(r_i,t)$, since all other terms in the sum are zero, when ∇_i is applied to a term independent of r_i. The resulting equation is:

$$I_2 = N\sum_i^N\int_\Omega\nabla_i\cdot\Bigg(P(r^N,t)\nabla_i V_{ext}(r_i,t)\Bigg)dr_2...dr_N.$$

As before, the Divergence Theorem can be used for all variables $r_2,...r_N$, while the equation for r_1 remains unchanged. This gives:

$$I_2 = N\sum_{i=2}^N\int_{\partial\Omega}P(r^N,t)\frac{\partial V_{ext}(r_i,t)}{\partial n}dr_2...dr_N+N\int_\Omega\nabla_1\cdot\Bigg(P(r^N,t)\nabla_1 V_{ext}(r_i,t)\Bigg)dr_2...dr_N,$$

where n is again the outward normal. Then, applying the boundary conditions for $P(r^N,t)$, as discussed above, and realising that $\nabla_1 V_{ext}(r_i,t) = \nabla_1 V_{ext}(r_1,t)$, since this expression is zero for all $r_i \neq r_1$, the following equations is derived:

$$I_2 = N\int_\Omega\nabla_1\cdot\Bigg(P(r^N,t)\nabla_1 V_{ext}(r_1,t)\Bigg)dr_2...dr_N.$$

Since r_1 is constant with respect to the integration variables, all terms only depending on r_1 can be taken out of the integral to give:

$$\begin{aligned} I_2 &= N\nabla_1 \cdot \left(\nabla_1 V_{ext}(r_1, t) \int_\Omega P(r^N, t) dr_2 ... dr_N \right) \\ &= \nabla_1 \cdot \left((\nabla_1 V_{ext}(r_1, t)) \rho(r, t) \right). \end{aligned}$$

Then, dropping the subscripts, I_2 is:

$$I_2 = \nabla \cdot \left(\rho(r, t) \nabla V_{ext}(r_1, t) \right). \tag{6}$$

The final term in the PDE that has to be considered is I_3:

$$I_3 = N \int_\Omega \sum_i^N \nabla_i \cdot \left(P(r^N, t) \nabla_i \frac{1}{2} \sum_{m \neq n}^N V_2(|r_m - r_n|) \right) dr_2 ... dr_N.$$

As for I_1 and I_2, the integration and summation operations can be swapped, and the Divergence Theorem can be applied to $r_2, ...r_N$ to give:

$$\begin{aligned} I_3 =& \frac{1}{2} N \sum_i^N \int_\Omega \nabla_i \cdot \left(P(r^N, t) \nabla_i \sum_{m \neq n}^N V_2(|r_m - r_n|) \right) dr_2 ... dr_N \\ =& \frac{1}{2} N \int_{\partial\Omega} \sum_{i=2}^N \left(P(r^N, t) \sum_{m \neq n}^N \frac{\partial V_2(|r_m - r_n|)}{\partial n} \right) dr_2 ... dr_N \\ &+ \frac{1}{2} N \int_\Omega \nabla_1 \cdot \left(P(r^N, t) \nabla_1 \sum_{m \neq n}^N V_2(|r_m - r_n|) \right) dr_2 ... dr_N. \end{aligned}$$

The boundary conditions for P are applied to set the first term to zero.
The term $\nabla_1 \sum_{m \neq n}^N V_2(|r_m - r_n|)$ has to be examined in more detail. Since the gradient is applied with respect to r_1, one of the r_m, r_n in the double sum has to be r_1, since all other terms will be zero when the gradient is applied. Therefore:

$$\nabla_1 \sum_{m \neq n}^N V_2(|r_m - r_n|) = \nabla_1 \sum_{n=2}^N V_2(|r_1 - r_n|) + \nabla_1 \sum_{m=2}^N V_2(|r_m - r_1|).$$

Since m and n are dummy variables and $|r_m - r_n| = |r_n - r_m|$ is symmetric, the previous equation reduces to:

$$\nabla_1 \sum_{m \neq n}^N V_2(|r_m - r_n|) = 2\nabla_1 \sum_{n=2}^N V_2(|r_1 - r_n|).$$

Then I_3 becomes:

$$I_3 = N \int_\Omega \nabla_1 \cdot \left(P(r^N, t) \nabla_1 \sum_{n=2}^N V_2(|r_1 - r_n|) \right) dr_2 ... dr_N.$$

Writing out the sum in N explicitly gives:

$$\begin{aligned} I_3 =& N \int_\Omega \nabla_1 \cdot \Big(P(r^N, t) \nabla_1 V_2(|r_1 - r_2|) \Big) dr_2 ... dr_N \\ &+ N \int_\Omega \nabla_1 \cdot \Big(P(r^N, t) \nabla_1 V_2(|r_1 - r_3|) \Big) dr_2 ... dr_N \\ &\vdots \\ &+ N \int_\Omega \nabla_1 \cdot \Big(P(r^N, t) \nabla_1 V_2(|r_1 - r_N|) \Big) dr_2 ... dr_N. \end{aligned}$$

Since the particles are indistinguishable, a permutation argument can be employed and the indices r_i in the terms $V_2(|r_1 - r_i|)$ can be relabelled, such that $r_i = r_2$, $i = 3, ..., n$, for each term in the sum. Since the integration is symmetric, the integration order can be permuted arbitrarily and hence does not have to be adapted. This results in $N - 1$ identical equations, and so I_3 is:

$$I_3 = N(N-1) \int_\Omega \nabla_1 \cdot \Big(P(r^N, t) \nabla_1 V_2(|r_1 - r_2|) \Big) dr_2 ... dr_N.$$

Considering now the definition of the n-body particle distributions, I_3 can be written in terms of the two-body distribution $\rho^{(2)}(r_1, r_2, t) = N(N-1) \int_\Omega dr_3 ... dr_N P(r^N, t)$. Terms that only depend on r_1 can be taken out of the integral. Then I_3 is:

$$I_3 = N(N-1) \nabla_1 \cdot \Big(\Big(\nabla_1 \int_\Omega V_2(|r_1 - r_2|) \Big) P(r^N, t) \Big) dr_2 ... dr_N.$$

Since V_2 only depends on r_2 and the order of integration is arbitrary, the integral can be rewritten as follows:

$$\begin{aligned} I_3 =& \nabla_1 \cdot \Big(\nabla_1 \int_\Omega V_2(|r_1 - r_2|) \Big(N(N-1) \int_\Omega P(r^N, t) dr_3 ... dr_N \Big) dr_2 \Big) \\ =& \nabla_1 \cdot \Big(\nabla_1 \int_\Omega V_2(|r_1 - r_2|) \rho^{(2)}(r_1, r_2, t) dr_2 \Big). \end{aligned}$$

Dropping the indices on ∇_1, the equation is:

$$I_3 = \nabla \cdot \Big(\nabla \int_\Omega V_2(|r - r_2|) \rho^{(2)}(r, r_2, t) dr_2 \Big). \tag{7}$$

The full three dimensional approximation of (4) is found by combining (5), (6) and (7), to give:

$$\partial_t \rho(r, t) = \nabla \cdot \Big(\nabla \rho(r, t) + \rho(r, t) \nabla V_{ext}(r_1, t) + \int_\Omega \nabla V_2(|r - r_2|) \rho^{(2)}(r, r_2, t) dr_2 \Big).$$

This equation is not closed, since it depends on $\rho^{(2)}(r, r_2, t)$. There are different approaches to closing the equation, and two of them are discussed below.

Note that the derivation for **w** term follows the same steps as above and is therefore omitted.

2.2 Approximating the Two-Body Density

There are different ways to approximate the two-body density $\rho^{(2)}$. In this section, two of these are presented; the Mean Field Approximation and Density Functional Theory.

2.2.1 The Mean Field Approximation

In order to use a mean field approach, a modelling assumption has to be made. It is assumed that the particles in the bath are only weakly interacting and the resulting approximation is that of independence. It is assumed that the two-body density of particles r_1 and r_2 is approximately the product of the individual one-body densities of r_1 and r_2, that is:

$$\rho^{(2)}(r, r_2, t) \approx \rho(r_1, t)\rho(r_2, t),$$

as in [6]. The resulting closed PDE is:

$$\partial_t \rho(r,t) = \nabla \cdot \left(\left(\nabla + \nabla V_{ext}(r_1, t) + \int_\Omega \nabla V_2(|r - r_2|)\rho(r_2, t) dr_2 \right) \rho(r,t) \right). \tag{8}$$

In the context of the mean field approximation, the integral term can be interpreted as the sum of forces between a particle at a position r and all other particles in Ω. However, the independence approximation is often not practical in modelling industrial phenomena and so an alternative approach needs to be considered, which includes the effects of the two-body density.

2.2.2 The Adiabatic Approximation

Another approach for approximating $\rho^{(2)}$ is using Density Functional Theory (DFT). DFT shows that at equilibrium, when $\partial_t \rho = 0$, there exists a free energy functional $\mathcal{F}$, such that:

$$\begin{aligned} &\nabla\rho(r,t) + \rho(r,t)\nabla V_{ext}(r_1, t) + \int_\Omega \nabla V_2(|r - r_2|)\rho^{(2)}(r, r_2, t) dr_2 \\ &= \rho(r,t)\nabla \frac{\delta \mathcal{F}[\rho]}{\delta \rho}, \end{aligned}$$

where $\frac{\delta}{\delta \rho}$ is the functional derivative with respect to ρ. While in most cases $\mathcal{F}$ is not known explicitly, it is known that at equilibrium $\nabla \frac{\delta \mathcal{F}[\rho]}{\delta \rho} = 0$, and therefore it can be assumed that the PDE can be rewritten as:

$$\partial_t \rho(r,t) = \nabla \cdot \left(\rho(r,t) \nabla \frac{\delta \mathcal{F}[\rho]}{\delta \rho} \right), \tag{9}$$

as discussed in [8]. This is called the Adiabatic Approximation, and $\mathcal{F}$ contains all of the information about particle correlations, if it is known. For non-interacting particles, the explicit form for $\mathcal{F}[\rho]$ is known to be:

$$\mathcal{F}[\rho] = \int \rho(\log(\rho) - 1) dr, \tag{10}$$

see [7].

To show that $\mathcal{F}[\rho]$ satisfies the adiabatic approximation (9), the functional derivative can be computed and substituted into (9). Since $\mathcal{F}$ is of the form $\mathcal{F}[\rho] = \int f(r, \rho(r), \nabla\rho(r))dr$, where $f(r, \rho(r), \nabla\rho(r)) = \rho(\log(\rho) - 1)$, a function ϕ of compact support can be defined, as discussed in [9], such that:

$$\begin{aligned}\int \frac{\delta\mathcal{F}[\rho]}{\delta\rho}\phi(r)dr &= \left[\frac{d}{d\epsilon}\int f(r, \rho(r) + \epsilon\phi, \nabla\rho(r) + \epsilon\nabla\phi)dr\right]_{\epsilon=0} \\ &= \int\left(\frac{\partial f}{\partial\rho} - \left(\nabla\cdot\frac{\partial f}{\partial\nabla\rho}\right)\right)\phi(r)dr.\end{aligned}$$

Since this holds for all functions $\phi \in C_0^\infty(\Omega)$, the following holds:

$$\frac{\delta\mathcal{F}[\rho]}{\delta\rho} = \left(\frac{\partial f}{\partial\rho} - \left(\nabla\cdot\frac{\partial f}{\partial\nabla\rho}\right)\right).$$

Applying this result to (10) results in:

$$\begin{aligned}\frac{\delta\mathcal{F}[\rho]}{\delta\rho} &= \frac{\delta}{\delta\rho}\left(\int\rho(\log(\rho) - 1)dr\right) \\ &= \frac{\partial}{\partial\rho}(\rho(\log(\rho) - 1)) - \nabla\cdot\left(\frac{\partial}{\partial\nabla\rho}(\rho(\log(\rho) - 1)\right) \\ &= \frac{\partial}{\partial\rho}(\rho(\log(\rho) - 1)) \\ &= \log\rho.\end{aligned}$$

Applying the gradient to this result gives:

$$\nabla\frac{\delta\mathcal{F}[\rho]}{\delta\rho} = \nabla\log\rho = \frac{\nabla\rho}{\rho}.$$

Substituting this into (9) results in:

$$\begin{aligned}\partial_t\rho(r,t) &= \nabla\cdot\left(\rho(r,t)\nabla\frac{\delta\mathcal{F}[\rho]}{\delta\rho}\right) \\ &= \nabla\cdot\left(\rho(r,t)\frac{\nabla\rho(r,t)}{\rho(r,t)}\right) \\ &= \nabla\cdot\nabla\rho(r,t) \\ &= \Delta\rho(r,t).\end{aligned}$$

This shows that the particular choice of $\mathcal{F}$, defined by (10), recovers the diffusion equation when substituted into (9). This is to be expected, since $\mathcal{F}$ represents non-interacting particles.

3 PDE-Constrained Optimization

The aim of this project is to work towards using the particle model derived in the previous section to describe an industrial process and optimize this process with minimal cost involved. It is of interest to achieve a particle distribution $\hat{\rho}$ in some time over some domain Ω. In the context of PDE-constrained optimization, the aim is to minimize the distance between a state variable ρ and a desired state $\hat{\rho}$, in some norm, while also minimizing the cost involved in reaching the desired state. This minimization is constrained by the underlying physics of the particle system. The PDE describing the particle dynamics is called the state equation.
Achieving the desired state $\hat{\rho}$ as close as possible can be of interest either for all times, as in this report, or only at some times, such as the final time T. In order to achieve $\hat{\rho}$, the particle distribution ρ can be controlled through a so-called control variable, denoted by u. The control can be applied in various ways, which is dependent on the application. Since the background flow influences the particle distribution ρ, $\hat{\rho}$ can try to be reached by changing the flow field. Then the flow field is the control u. Alternatively, u could represent the temperature or the geometry of the boundaries of the bath. Moreover, u could be a parameter in the body force or in the particle interaction term, influencing the particle distribution through the forces involved. Note that u cannot always influence the system enough to reach the desired state $\hat{\rho}$. This highly depends on the choice of $\hat{\rho}$, the physics of the problem and on the choice of the parameter β, which is discussed below. Since controlling ρ requires energy, u can be thought of as the cost involved in reaching $\hat{\rho}$.
The weight of the control is determined by the so-called regularization parameter, β. If β is small, the desired state will be reached, however, at a high cost. If β is large, the control will be minimized, but the desired state might not be reached. The choice of β depends on the application involved. It is generally of interest to find a range of β values, for which the solution to the optimization problem is robust. The PDE-constrained optimization problem of interest in this report is of the form:

$$\min_{\rho,\mathbf{w}} \quad \frac{1}{2}\|\rho - \hat{\rho}\|_{L^2}^2 + \frac{\beta}{2}\|\mathbf{w}\|_{L^2}^2 \,, \tag{11}$$

subject to:

$$\partial_t \rho = \nabla^2 \rho - \nabla \cdot (\rho \mathbf{w}) + \nabla \cdot (\rho \nabla V_{ext}) \quad \text{in} \quad \Omega,$$

$$\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n} = 0 \quad \text{on} \quad \partial\Omega,$$

$$\rho = \rho_0 \quad \text{at} \quad t = 0,$$

where the state equation is a simplified version of (1), which neglects the particle interaction term. Note that the norms are the L^2 norms with respect to Ω and time. Other norms could be used, depending on the type of application. In the following it is assumed, as described in [10], that $\rho \in H^1$ and $\mathbf{w} \in L^2$. In order to solve this optimization problem, continuous optimality conditions can be derived, which can then be discretized and solved numerically. This is known as the optimize-then-discretize approach. Another approach, discretize-then-optimize, would be to first discretize (11) and then derive the discrete optimality conditions that need to be solved. A good introduction to PDE-constrained optimization, can be found in [11] and a more detailed introduction to numerical PDE-constrained optimization is provided in [12]. Both of these texts provided the basis for the above discussion.

3.1 Deriving First-Order Optimality Conditions

Employing the optimize-then-discretize approach, the continuous first order optimality conditions are derived using a Lagrangian approach. These are necessary optimality conditions, however, the sufficient second-order optimality conditions would be needed to ensure that the stationary points found via the first-order optimality conditions are indeed the minima of the problem. Sufficient optimality conditions are not part of this report but can be found, along the underlying theory of the approach employed in this section, in [12] and [10]. The derivation of the first-order optimality system follows closely the presentation in [10]. The PDE-constrained optimization problem (11) in Lagrangian form is:

$$\begin{aligned}\mathcal{L}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega}) = &\frac{1}{2}\|\rho - \hat{\rho}\|^2_{L^2(\Omega,t)} + \frac{\beta}{2}\|\mathbf{w}\|^2_{L^2(\Omega,t)} \\ &+ \int_0^T \int_\Omega \Big(\partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho \mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext})\Big) p_\Omega dr dt \\ &+ \int_0^T \int_{\partial\Omega} \left(\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n}\right) p_{\partial\Omega} dr dt,\end{aligned}$$

where p_Ω and $p_{\partial\Omega}$ are the Lagrange multipliers of the problem, relating to the interior and the boundary of Ω, respectively. The Lagrange multiplier $p_\Omega \in L^2$ is called the adjoint variable in the resulting system of equations. Writing the L^2 norms explicitly gives:

$$\begin{aligned}\mathcal{L}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega}) = &\frac{1}{2}\int_0^T \int_\Omega (\rho - \hat{\rho})^2 dr dt + \frac{\beta}{2}\int_0^T \int_\Omega \mathbf{w}^2 dr dt \\ &+ \int_0^T \int_\Omega \Big(\partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho \mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext})\Big) p_\Omega dr dt \\ &+ \int_0^T \int_{\partial\Omega} \left(\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n}\right) p_{\partial\Omega} dr dt.\end{aligned} \tag{12}$$

It is beneficial to rewrite the part of (12) that comes from the state equation, in order to simplify further computations. The term of interest is:

$$\int_0^T \int_\Omega \Big(\partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho \mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext}) \Big) p_\Omega dr dt$$
$$= \int_0^T \int_\Omega p_\Omega \partial_t \rho dr dt - \int_0^T \int_\Omega p_\Omega \nabla^2 \rho dr dt + \int_0^T \int_\Omega \nabla \cdot (\rho \mathbf{w}) p_\Omega dr dt - \int_0^T \int_\Omega \nabla \cdot (\rho \nabla V_{ext}) p_\Omega dr dt$$
$$:= I_1 + I_2 + I_3 + I_4.$$

Each of the defined integrals is considered in turn. The first integral contains the time derivative of ρ. Integration by parts is applied to get:

$$\begin{aligned} I_1 &= \int_0^T \int_\Omega p_\Omega \partial_t \rho dr dt = \Big[\int_\Omega \rho p_\Omega dr \Big]_0^T - \int_0^T \int_\Omega \rho \partial_t p_\Omega dr dt \\ &= \int_\Omega (\rho(T) p_\Omega(T) - \rho_0 p_\Omega(0)) dr - \int_0^T \int_\Omega \rho \partial_t p_\Omega dr dt. \end{aligned}$$

In order to rewrite the second integral, integration by parts is applied twice as follows:

$$\begin{aligned} I_2 &= \int_0^T \int_\Omega p_\Omega \nabla^2 \rho dr dt = \int_0^T \int_{\partial\Omega} p_\Omega \frac{\partial \rho}{\partial n} dr dt - \int_0^T \int_\Omega \nabla \rho \cdot \nabla p_\Omega dr dt \\ &= \int_0^T \int_{\partial\Omega} p_\Omega \frac{\partial \rho}{\partial n} dr dt - \int_0^T \int_{\partial\Omega} \rho \frac{\partial p_\Omega}{\partial n} dr dt + \int_0^T \int_\Omega \rho \nabla^2 p_\Omega dr dt. \end{aligned}$$

The third integral is rewritten using similar arguments:

$$I_3 = \int_0^T \int_\Omega \nabla \cdot (\rho w) p_\Omega dr dt = \int_0^T \int_{\partial\Omega} p_\Omega \rho w \cdot n dr dt - \int_0^T \int_\Omega \rho \nabla p_\Omega \cdot w dr dt.$$

The final integral is rewritten as follows:

$$\begin{aligned} I_4 &= \int_0^T \int_\Omega \nabla \cdot (\rho \nabla V_{ext}) p_\Omega dr dt = \int_0^T \int_\Omega \nabla \cdot (\rho \nabla V_{ext} p_\Omega) dr dt - \int_0^T \int_\Omega \rho \nabla p_\Omega \cdot \nabla V_{ext} dr dt \\ &= \int_0^T \int_{\partial\Omega} \rho p_\Omega \frac{\partial V_{ext}}{\partial n} dr dt - \int_0^T \int_\Omega \rho \nabla p_\Omega \cdot \nabla V_{ext} dr dt, \end{aligned}$$

where the Divergence Theorem is used to derive the final result. Substituting the four rewritten integrals back into (12) results in:

$$\begin{aligned} \mathcal{L}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega}) = {} & \frac{1}{2} \int_0^T \int_\Omega (\rho - \hat{\rho})^2 dr dt + \frac{\beta}{2} \int_0^T \int_\Omega \mathbf{w}^2 dr dt + \int_\Omega (\rho(T) p_\Omega(T) - \rho_0 p_\Omega(0)) dr \\ & - \int_0^T \int_\Omega \rho \partial_t p_\Omega dr dt - \int_0^T \int_{\partial\Omega} p_\Omega \frac{\partial \rho}{\partial n} dr dt + \int_0^T \int_{\partial\Omega} \rho \frac{\partial p_\Omega}{\partial n} dr dt - \int_0^T \int_\Omega \rho \nabla^2 p_\Omega dr dt \\ & + \int_0^T \int_{\partial\Omega} p_\Omega \rho \mathbf{w} \cdot \mathbf{n} dr dt - \int_0^T \int_\Omega \rho \nabla p_\Omega \cdot \mathbf{w} dr dt - \int_0^T \int_{\partial\Omega} \rho p_\Omega \frac{\partial V_{ext}}{\partial n} dr dt \\ & + \int_0^T \int_\Omega \rho \nabla p_\Omega \cdot \nabla V_{ext} dr dt + \int_0^T \int_{\partial\Omega} \Big(\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n} \Big) p_{\partial\Omega} dr dt. \end{aligned}$$

Sorting the terms by whether they are interior or boundary terms and grouping them gives:

$$\mathcal{L}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega}) = \int_\Omega (\rho(T)p_\Omega(T) - \rho_0 p_\Omega(0))dr \tag{13}$$
$$+ \int_0^T \int_\Omega \left(\frac{1}{2}(\rho - \hat{\rho})^2 + \frac{\beta}{2}\mathbf{w}^2 - \rho\partial_t p_\Omega - \rho\nabla p_\Omega \cdot \mathbf{w} - \rho\nabla^2 p_\Omega + \rho\nabla p_\Omega \cdot \nabla V_{ext}\right) drdt$$
$$+ \int_0^T \int_{\partial\Omega} \left(\rho\frac{\partial p_\Omega}{\partial n} - \rho p_\Omega \frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial \rho}{\partial n} + p_\Omega \rho \mathbf{w} \cdot \mathbf{n} + p_{\partial\Omega}\frac{\partial \rho}{\partial n} - \rho p_{\partial\Omega}\mathbf{w} \cdot \mathbf{n} + \rho p_{\partial\Omega}\frac{\partial V_{ext}}{\partial n}\right) drdt.$$

In order to derive the first-order optimality conditions for the optimization problem, the Fréchet derivatives of $\mathcal{L}$ with respect to all variables $\rho, w, p_\Omega, p_{\partial\Omega}$ have to be taken. The system of equations that results from setting these first derivatives to zero are called first-order optimality conditions. The resulting equations are called the adjoint equation, the gradient equation and the forward problem. The latter is equivalent to the state equation.

3.1.1 Fréchet Differentiation

In the derivation of the optimality conditions, Fréchet derivatives of the Lagrangian have to be taken. In order to do so, the notions of derivatives needed in this context are introduced in this section.
The following definitions are taken from [12] and [13]. Throughout this section, let U, V and Z be real Banach spaces.

Definition 3.1. [12] If, for given elements $u, h \in U$, the limit

$$\delta F(u)(h) := \lim_{t \to 0^+} \frac{1}{t}\Big(F(u + th) - F(u)\Big)$$

exists, then $\delta F(u)(h)$ is called the *directional derivative* of F at u in direction h. If this limit exists for all $h \in U$, then F is called directionally differentiable at u.

Definition 3.2. [12] If, for some $u \in U$ and all $h \in U$ the limit

$$\delta F(u)(h) := \lim_{t \to 0} \frac{1}{t}\Big(F(u + th) - F(u)\Big)$$

exists and $\delta F(u)(h)$ is a continuous mapping from U to V, then $\delta F(u)$ is denoted by $F'(u)$ and is called the Gâteaux derivative of F at u, and F is called *Gâteaux differentiable* at u.

Definition 3.3. [12] If F is Gâteaux differentiable at $u \in U$, and satisfies in addition that

$$\lim_{\|h\|_U \to 0} \frac{\big\|F(u + h) - F(u) - F'(u)h\big\|_V}{\|h\|_U} = 0,$$

then $F'(u)$ is called the Fréchet derivative of F at u and F is called *Fréchet differentiable*.

Definition 3.4. [12] *Chain Rule*
Let $F : U \to V$ and $G : V \to Z$ be Fréchet differentiable at u and $F(u)$, respectively. Then

$$E(u) = G(F(u))$$

is also Fréchet differentiable and its derivative is given by:

$$E'(u) = G'(F(u))F'(u).$$

Definition 3.5. [13] *Product Rule*
Let U and V be Banach spaces, let $\mathcal{U}$ be an open subset of U, and let $F : \mathcal{U} \to \mathbf{R}$ and $G : \mathcal{U} \to V$ be functions. If both F and G are differentiable at $u \in \mathcal{U}$, then FG is differentiable at u and

$$(FG)'(u) = F(u)G'(u) + G(u)F'(u).$$

3.1.2 Deriving the Adjoint Equation

The adjoint equation is found by calculating the Fréchet derivative of the Lagrangian (13) with respect to the state variable and setting it equal to zero. The derivative with respect to ρ is:

$$\begin{aligned}
&\mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h = \int_\Omega h(T)p_\Omega(T)dr \\
&+ \int_0^T \int_\Omega \Big((\rho - \hat{\rho})h - h\partial_t p_\Omega - h\nabla p_\Omega \cdot \mathbf{w} - h\nabla^2 p_\Omega + h\nabla p_\Omega \cdot \nabla V_{ext}\Big)drdt \\
&+ \int_0^T \int_{\partial\Omega} \Big(h\frac{\partial p_\Omega}{\partial n} - hp_\Omega\frac{\partial V_{ext}}{\partial n} - p_\Omega\frac{\partial h}{\partial n} + p_\Omega h\mathbf{w}\cdot\mathbf{n} + p_{\partial\Omega}\frac{\partial h}{\partial n} - hp_{\partial\Omega}\mathbf{w}\cdot\mathbf{n} + hp_{\partial\Omega}\frac{\partial V_{ext}}{\partial n}\Big)drdt \\
&= \int_\Omega h(T)p_\Omega(T)dr + \int_0^T \int_\Omega \Big((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext}\Big)hdrdt \\
&+ \int_0^T \int_{\partial\Omega} \Big(\Big(\frac{\partial p_\Omega}{\partial n} + p_\Omega\mathbf{w}\cdot\mathbf{n} - p_{\partial\Omega}\mathbf{w}\cdot\mathbf{n} + p_{\partial\Omega}\frac{\partial V_{ext}}{\partial n} - p_\Omega\frac{\partial V_{ext}}{\partial n}\Big)h + \Big(p_{\partial\Omega} - p_\Omega\Big)\frac{\partial h}{\partial n}\Big)drdt,
\end{aligned}$$

where $h \in L^2$ belongs to the same space as ρ and can be thought of as the direction of differentiation. The initial condition for ρ, ρ_0, vanishes from the derivative of $\mathcal{L}$, because h satisfies $h(r, 0) = 0$. As discussed in [10], this is because the variational inequality $\mathcal{L}_\rho(\tilde{\rho}, \tilde{\mathbf{w}}, p_\Omega, p_{\partial\Omega})(\rho - \tilde{\rho}) \geq 0$ has to be satisfied for all admissible ρ, in order for $\tilde{\rho}$ and $\tilde{\mathbf{w}}$ to be the minimum of the problem. If $h := \rho - \tilde{\rho}$, then $h(r, 0) = 0$ and furthermore, $-h$ is also an admissible choice of function. Therefore, the variational inequality becomes the equality $\mathcal{L}_\rho(\tilde{\rho}, \tilde{\mathbf{w}}, p_\Omega, p_{\partial\Omega})h = 0$, for h sufficiently smooth, with $h(r, 0) = 0$.
Setting $\mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h = 0$, in order to find the adjoint equation, gives:

$$\begin{aligned}
&\mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h = \int_\Omega h(T)p_\Omega(T)dr \qquad (14)\\
&+ \int_0^T \int_\Omega \Big((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext}\Big)hdrdt \\
&+ \int_0^T \int_{\partial\Omega} \Big(\Big(\frac{\partial p_\Omega}{\partial n} + p_\Omega\mathbf{w}\cdot\mathbf{n} - p_{\partial\Omega}\mathbf{w}\cdot\mathbf{n} + p_{\partial\Omega}\frac{\partial V_{ext}}{\partial n} - p_\Omega\frac{\partial V_{ext}}{\partial n}\Big)h + \Big(p_{\partial\Omega} - p_\Omega\Big)\frac{\partial h}{\partial n}\Big)drdt = 0.
\end{aligned}$$

The first step is to restrict the choice of h as much as possible. That is choosing $h \in C_0^\infty(\Omega)$, such that:

$$\begin{aligned} h(T) = 0 \quad &\text{in} \quad \Omega, \\ h = 0 \quad &\text{on} \quad \partial\Omega, \\ \frac{\partial h}{\partial n} = 0 \quad &\text{on} \quad \partial\Omega, \end{aligned} \tag{15}$$

as discussed in [10]. With this choice of h, (14) reduces to:

$$\begin{aligned} \mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h &= \int_0^T \int_\Omega \Big((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} \Big) h dr dt \\ &= 0. \end{aligned} \tag{16}$$

Since this equation has to hold for all h satisfying (15) and, as discussed in [10], $C_0^\infty(\Omega)$ is dense in $L^2(\Omega)$, it can be concluded that:

$$(\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} = 0.$$

The adjoint equation is then

$$-\partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} = -(\rho - \hat{\rho}).$$

Now it is of interest to relax the conditions on h, to derive the results at the boundary $\partial\Omega$ and at the final time T. First, the case where $h(T) \neq 0$ is considered, so that $h \in C^1(\Omega)$. The remaining conditions are:

$$\begin{aligned} h = 0 \quad &\text{on} \quad \partial\Omega, \\ \frac{\partial h}{\partial n} = 0 \quad &\text{on} \quad \partial\Omega. \end{aligned} \tag{17}$$

Therefore, (14) reduces to:

$$\begin{aligned} \mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h = &\int_\Omega h(T) p_\Omega(T) dr \\ &+ \int_0^T \int_\Omega \Big((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} \Big) h dr dt = 0. \end{aligned}$$

However, since $\int_0^T \int_\Omega \Big((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} \Big) h dr dt = 0$ by (16), the equation reduces to:

$$\int_\Omega h(T) p_\Omega(T) dr = 0, \tag{18}$$

for all $h(T) \in \Omega$, and so:

$$p_\Omega(r, T) = 0 \quad \text{in} \quad \Omega, \tag{19}$$

by the same density argument of the spaces involved, see [10]. In order to further ease the requirements on h, $\frac{\partial h}{\partial n} \neq 0$ on $\partial\Omega$ is permitted and the only remaining restriction on $h \in C^1(\Omega)$ is:

$$h = 0 \quad \text{on} \quad \partial\Omega.$$

Then (14) reduces to:

$$\begin{aligned}\mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h = &\int_\Omega h(T)p_\Omega(T)dr \\ &+ \int_0^T \int_\Omega \Big((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext}\Big) h dr dt \\ &+ \int_0^T \int_{\partial\Omega} \Big(p_{\partial\Omega} - p_\Omega\Big) \frac{\partial h}{\partial n} dr dt = 0.\end{aligned}$$

Since the first two terms vanish by (16) and (18), this reduces to:

$$\int_0^T \int_{\partial\Omega} \Big(p_{\partial\Omega} - p_\Omega\Big) \frac{\partial h}{\partial n} dr dt = 0. \tag{20}$$

This holds for all permissible choices of h and the set of these h is dense in $L^2(\Omega)$. Therefore, as before, this concludes that:

$$p_{\partial\Omega} = p_\Omega. \tag{21}$$

This provides a relationship between the two Lagrange multipliers and therefore only p_Ω remains in the final adjoint equation as the so-called adjoint variable. Finally, all restrictions on h are lifted and $h \neq 0$ on $\partial\Omega$ is a permitted choice. Since all other terms in (14) vanish by (16), (18) and (20), it reduces to:

$$\int_0^T \int_{\partial\Omega} \left(\left(\frac{\partial p_\Omega}{\partial n} + p_\Omega \mathbf{w} \cdot \mathbf{n} - p_{\partial\Omega} \mathbf{w} \cdot \mathbf{n} + p_{\partial\Omega} \frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial V_{ext}}{\partial n} \right) h dr dt = 0. \right.$$

This has to hold for all h and by the same density argument as before:

$$\frac{\partial p_\Omega}{\partial n} + p_\Omega \mathbf{w} \cdot \mathbf{n} - p_{\partial\Omega} \mathbf{w} \cdot \mathbf{n} + p_{\partial\Omega} \frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial V_{ext}}{\partial n} = 0.$$

Now, using (21), this is:

$$\frac{\partial p_\Omega}{\partial n} + p_\Omega \mathbf{w} \cdot \mathbf{n} - p_\Omega \mathbf{w} \cdot \mathbf{n} + p_\Omega \frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial V_{ext}}{\partial n} = 0,$$

and the boundary condition for the adjoint equation reduces to:

$$\frac{\partial p_\Omega}{\partial n} = 0. \tag{22}$$

Finally, the adjoint equation, including boundary and final time conditions, satisfies:

$$\begin{aligned} -\partial_t p - \nabla p \cdot \mathbf{w} - \nabla^2 p + \nabla p \cdot \nabla V_{ext} &= -(\rho - \hat{\rho}) \quad \text{in} \quad \Omega, \\ p(T) &= 0 \quad \text{in} \quad \Omega, \\ \frac{\partial p}{\partial n} &= 0 \quad \text{on} \quad \partial\Omega, \end{aligned} \tag{23}$$

where $p := p_\Omega$ for notational convenience.

3.1.3 The First-Order Optimality System

The gradient equation is derived by taking the Fréchet derivative of (13) with respect to $\mathbf{w}$ and setting it equal to zero. The derivative satisfies:

$$\begin{aligned} \mathcal{L}_{\mathbf{w}}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})\mathbf{h} &= \int_0^T \int_\Omega \Big(\beta \mathbf{w} \cdot \mathbf{h} - \rho \nabla p_\Omega \cdot \mathbf{h}\Big) dr dt + \int_0^T \int_{\partial\Omega} \Big(p_\Omega \rho \mathbf{h} \cdot \mathbf{n} - p_{\partial\Omega} \rho \mathbf{h} \cdot \mathbf{n}\Big) dr dt \\ &= 0. \end{aligned}$$

Using (21), the boundary terms cancel and the equation reduces to:

$$\int_0^T \int_\Omega \Big(\beta \mathbf{w} - \rho \nabla p_\Omega\Big) \cdot \mathbf{h} dr dt = 0.$$

Since this has to hold for all choices of $\mathbf{h} \in L^2(\Omega)$, this gives the gradient equation:

$$\beta \mathbf{w} - \rho \nabla p_\Omega = 0 \quad \text{in} \quad \Omega. \tag{24}$$

The state equation is derived by taking the Fréchet derivative of (12) with respect to p_Ω, and setting the result equal to zero. Note that this result makes use of (21). The derivative is:

$$\begin{aligned} \mathcal{L}_{p_\Omega}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h &= \int_0^T \int_\Omega \Big(\partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho \mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext})\Big) h dr dt \\ &+ \int_0^T \int_{\partial\Omega} \Big(\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n}\Big) h dr dt = 0. \end{aligned}$$

By the same argument as in the previous section, the forward equation, including the boundary conditions, is recovered:

$$\begin{aligned} \partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho \mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext}) &= 0 \quad \text{in} \quad \Omega, \\ \frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n} &= 0 \quad \text{on} \quad \partial\Omega. \end{aligned} \tag{25}$$

The system of the three equations that describe the first-order optimality conditions for the PDE-constrained optimization problem (11) consists of the adjoint and gradient equations as

well as the forward problem:

$$\begin{aligned}
&\textbf{Adjoint Equation} && \\
-\partial_t p - \nabla p \cdot \mathbf{w} - \nabla^2 p + \nabla p \cdot \nabla V_{ext} &= -(\rho - \hat{\rho}) && \text{in} \quad \Omega, \\
p(r,T) &= 0, && \\
\frac{\partial p}{\partial n} &= 0 && \text{on} \quad \partial\Omega, \\
&\textbf{Gradient Equation} && \\
\beta \mathbf{w} - \rho \nabla p_\Omega &= 0 && \text{in} \quad \Omega, \\
&\textbf{Forward Problem} && \\
\partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho \mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext}) &= 0 && \text{in} \quad \Omega, \\
\rho(r,0) &= \rho_0(r), && \\
\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n} &= 0 && \text{on} \quad \partial\Omega.
\end{aligned} \tag{26}$$

3.2 Adding a Non-Local Term

The PDE-constrained optimization problem can be extended by adding the non-local particle interaction term into the PDE constraint, as given by (8). The PDE-constrained optimization problem becomes:

$$\min_{\rho,\mathbf{w}} \quad \frac{1}{2}\|\rho - \hat{\rho}\|_{L^2}^2 + \frac{\beta}{2}\|\mathbf{w}\|_{L^2}^2 \,, \tag{27}$$

subject to:

$$\partial_t \rho = \nabla^2 \rho - \nabla \cdot (\rho \mathbf{w}) + \nabla \cdot (\rho \nabla V_{ext}) + \nabla \cdot \int_\Omega \rho(r)\rho(r')\nabla V_2(|r - r'|)dr' \quad \text{in} \quad \Omega,$$

$$\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{u} + \rho \frac{\partial V_{ext}}{\partial n} + \int_\Omega \rho(r)\rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' = 0 \qquad \text{on} \quad \partial\Omega,$$

$$\rho = \rho_0 \quad \text{at} \quad t = 0.$$

The Lagrangian of this problem follows directly from (12) in Section 3.1.2. The difference is the additional term from the non-local term in the PDE:

$$\begin{aligned}
\mathcal{L}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega}) &= \frac{1}{2}\int_0^T \int_\Omega (\rho - \hat{\rho})^2 dr dt + \frac{\beta}{2}\int_0^T \int_\Omega \mathbf{w}^2 dr dt \\
&+ \int_0^T \int_\Omega \left(\partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho \mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext}) - \nabla \cdot \int_\Omega \rho(r)\rho(r')\nabla V_2(|r - r'|)dr' \right) p_\Omega dr dt \\
&+ \int_0^T \int_{\partial\Omega} \left(\frac{\partial \rho}{\partial n} - \rho \mathbf{w} \cdot \mathbf{n} + \rho \frac{\partial V_{ext}}{\partial n} + \int_\Omega \rho(r)\rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' \right) p_{\partial\Omega} dr dt.
\end{aligned} \tag{28}$$

The non-local term over Ω is denoted by I_1 and rewritten as follows:

$$\begin{aligned}
I_1 &= \int_0^T \int_\Omega p_\Omega \nabla \cdot \Big(\int_\Omega \rho(r)\rho(r')\nabla V_2(|r-r'|)dr' \Big) dr dt \\
&= \int_0^T \int_\Omega p_\Omega \nabla \cdot \Big(\rho(r) \int_\Omega \rho(r')\nabla V_2(|r-r'|)dr' \Big) dr dt \\
&= \int_0^T \int_{\partial\Omega} p_\Omega \rho(r) \int_\Omega \rho(r') \frac{\partial V_2(|r-r'|)}{\partial n} dr' dr dt \\
&- \int_0^T \int_\Omega \nabla p_\Omega \Big(\rho(r) \int_\Omega \rho(r')\nabla V_2(|r-r'|)dr' \Big) dr dt,
\end{aligned}$$

by applying integration by parts. The non-local term, originating from the boundary condition, is denoted by I_2 and rewritten as:

$$\begin{aligned}
I_2 &= \int_0^T \int_{\partial\Omega} p_{\partial\Omega} \int_\Omega \rho(r)\rho(r') \frac{\partial V_2(|r-r'|)}{\partial n} dr' dr dt \\
&= \int_0^T \int_{\partial\Omega} p_{\partial\Omega} \rho(r) \int_\Omega \rho(r') \frac{\partial V_2(|r-r'|)}{\partial n} dr' dr dt.
\end{aligned}$$

The integrals I_1 and I_2 are rearranged and split up into integrals over Ω and over $\partial\Omega$. They are defined as follows:

$$I_\Omega = \int_0^T \int_\Omega \nabla p_\Omega \Big(\rho(r) \int_\Omega \rho(r')\nabla V_2(|r-r'|)dr' \Big) dr dt,$$

and

$$I_{\partial\Omega} = \int_0^T \int_{\partial\Omega} (p_{\partial\Omega} - p_\Omega)\rho(r) \int_\Omega \rho(r') \frac{\partial V_2(|r-r'|)}{\partial n} dr' dr dt.$$

The full Lagrangian is then similar to (13) in Section 3.1.2:

$$\begin{aligned}
&\mathcal{L}(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega}) = \int_\Omega \Big(\rho(T)p_\Omega(T) - \rho_0 p_\Omega(0) \Big) dr \\
&+ \int_0^T \int_\Omega \Big(\frac{1}{2}(\rho - \hat{\rho})^2 + \frac{\beta}{2}\mathbf{w}^2 - \rho\partial_t p_\Omega - \rho\nabla p_\Omega \cdot \mathbf{w} - \rho\nabla^2 p_\Omega + \rho\nabla p_\Omega \cdot \nabla V_{ext} \Big) dr dt \\
&+ \int_0^T \int_\Omega \nabla p_\Omega \Big(\rho(r) \int_\Omega \rho(r')\nabla V_2(|r-r'|)dr' \Big) dr dt \\
&+ \int_0^T \int_{\partial\Omega} (p_{\partial\Omega} - p_\Omega)\rho(r) \int_\Omega \rho(r') \frac{\partial V_2(|r-r'|)}{\partial n} dr' dr dt \\
&+ \int_0^T \int_{\partial\Omega} \Big(\rho \frac{\partial p_\Omega}{\partial n} - \rho p_\Omega \frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial \rho}{\partial n} + p_\Omega \rho \mathbf{w} \cdot \mathbf{n} + p_{\partial\Omega} \frac{\partial \rho}{\partial n} - \rho p_{\partial\Omega} \mathbf{w} \cdot \mathbf{n} + \rho p_{\partial\Omega} \frac{\partial V_{ext}}{\partial n} \Big) dr dt.
\end{aligned} \tag{29}$$

Equivalently to the procedure in Section 3.1.2, the Fréchet derivatives of the Lagrangian with respect to all variables have to be taken and set to zero. The derivatives of the non-local terms I_Ω and $I_{\partial\Omega}$ are taken separately and are then substituted into the result for the derivatives from the previous sections.

3.2.1 Deriving the Adjoint Equation

The Fréchet derivative of I_Ω with respect to ρ in direction h is:

$$(I_\Omega)_\rho(\rho, p_\Omega)h = \int_0^T \int_\Omega \nabla p_\Omega \bigg(h(r) \int_\Omega \rho(r') \nabla V_2(|r - r'|) dr' + \rho(r) \int_\Omega h(r') \nabla V_2(|r - r'|) dr' \bigg) dr dt,$$

using the product rule for Fréchet differentiation. For the term involving $h(r')$, the order of integration can be changed, as follows:

$$\begin{aligned}(I_\Omega)_\rho(\rho, p_\Omega)h = &\int_0^T \int_\Omega \nabla p_\Omega h(r) \int_\Omega \rho(r') \nabla V_2(|r - r'|) dr' dr dt \\ &+ \int_0^T \int_\Omega h(r') \int_\Omega \nabla p_\Omega \rho(r) \nabla V_2(|r - r'|) dr dr' dt.\end{aligned}$$

The variable names r and r' in the second integral are swapped for readability. This is possible since r and r' are dummy variables and V_2 is symmetric. Then $(I_\Omega)_\rho(\rho, p_\Omega)h$ can be rewritten as:

$$(I_\Omega)_\rho(\rho, p_\Omega)h = \int_0^T \int_\Omega h(r) \int_\Omega (\nabla p_\Omega(r) + \nabla p_\Omega(r')) \rho(r') \nabla V_2(|r - r'|) dr' dr dt.$$

The Fréchet derivative for $I_{\partial\Omega}$ with respect to ρ is:

$$\begin{aligned}(I_{\partial\Omega})_\rho(\rho, p_{\partial\Omega})h = \int_0^T \int_{\partial\Omega} (p_{\partial\Omega} - p_\Omega) \bigg(&h(r) \int_\Omega \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' \\ &+ \rho(r) \int_\Omega h(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' \bigg) dr dt,\end{aligned}$$

where again the product rule for Fréchet differentiation is applied. Furthermore, as for I_Ω, the order of integration is changed for the second term and the labelling of r and r' are swapped for convenience:

$$\begin{aligned}(I_{\partial\Omega})_\rho(\rho, p_{\partial\Omega})h = &\int_0^T \int_{\partial\Omega} (p_{\partial\Omega}(r) - p_\Omega(r)) h(r) \int_\Omega \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' dr dt \\ &+ \int_0^T \int_\Omega h(r) \int_{\partial\Omega} (p_{\partial\Omega}(r') - p_\Omega(r')) \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' dr dt.\end{aligned}$$

The derivatives for I_Ω and $I_{\partial\Omega}$ are combined with the derivatives for the other terms, as derived in (14), to give the full derivative of the Lagrangian defined in (29). This is set equal to zero,

as before, to derive the adjoint equation. The derivative is:

$$\begin{aligned}
&\mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h = \int_\Omega h(T)p_\Omega(T)dr \\
&+ \int_0^T \int_\Omega \Big((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} \Big) h dr dt \\
&+ \int_0^T \int_{\partial\Omega} \Bigg(\Big(\frac{\partial p_\Omega}{\partial n} + p_\Omega \mathbf{w} \cdot \mathbf{n} - p_{\partial\Omega} \mathbf{w} \cdot \mathbf{n} + p_{\partial\Omega} \frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial V_{ext}}{\partial n} \Big) h + \Big(p_{\partial\Omega} - p_\Omega \Big) \frac{\partial h}{\partial n} \Bigg) dr dt \\
&+ \int_0^T \int_\Omega h(r) \int_\Omega (\nabla p_\Omega(r) + \nabla p_\Omega(r')) \rho(r') \nabla V_2(|r - r'|) dr' dr dt \\
&+ \int_0^T \int_{\partial\Omega} (p_{\partial\Omega} - p_\Omega) h(r) \int_\Omega \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' dr dt \\
&+ \int_0^T \int_\Omega h(r) \int_{\partial\Omega} (p_{\partial\Omega}(r') - p_\Omega(r')) \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' dr dt = 0,
\end{aligned}$$

where again ρ_0 vanishes, due to the fact that $h(r, 0) = 0$, as discussed above and in [10]. The first three integral terms result from (14), the next one from differentiating I_Ω and the last two from differentiating $I_{\partial\Omega}$. Sorting the terms based on whether they are integral or boundary terms gives:

$$\begin{aligned}
&\mathcal{L}_\rho(\rho, \mathbf{w}, p_\Omega, p_{\partial\Omega})h = \int_\Omega h(T)p_\Omega(T)dr \\
&+ \int_0^T \int_\Omega \Bigg((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} \\
&+ \int_\Omega (\nabla p_\Omega(r) + \nabla p_\Omega(r')) \rho(r') \nabla V_2(|r - r'|) dr' + \int_{\partial\Omega} (p_{\partial\Omega}(r') - p_\Omega(r')) \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' \Bigg) h dr dt \\
&+ \int_0^T \int_{\partial\Omega} \Bigg(\Big(\frac{\partial p_\Omega}{\partial n} + p_\Omega \mathbf{w} \cdot \mathbf{n} - p_{\partial\Omega} \mathbf{w} \cdot \mathbf{n} + p_{\partial\Omega} \frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial V_{ext}}{\partial n} + (p_{\partial\Omega} - p_\Omega) \int_\Omega \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' \Big) h \\
&+ \Big(p_{\partial\Omega} - p_\Omega \Big) \frac{\partial h}{\partial n} \Bigg) dr dt = 0.
\end{aligned}$$

As before, $h \in C_0^\infty(\Omega)$ is restricted to satisfy:

$$\begin{aligned}
h(T) &= 0 \quad \text{in} \quad \Omega, \\
h &= 0 \quad \text{on} \quad \partial\Omega, \\
\frac{\partial h}{\partial n} &= 0 \quad \text{on} \quad \partial\Omega,
\end{aligned}$$

compare to (15). Then

$$\begin{aligned}
&\int_0^T \int_\Omega \Bigg((\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} \\
&+ \int_\Omega (\nabla p_\Omega(r) + \nabla p_\Omega(r')) \rho(r') \nabla V_2(|r - r'|) dr' \\
&+ \int_{\partial\Omega} (p_{\partial\Omega}(r') - p_\Omega(r')) \rho(r') \frac{\partial V_2(|r - r'|)}{\partial n} dr' \Bigg) h dr dt = 0,
\end{aligned}$$

which implies that:

$$
\begin{aligned}
&(\rho - \hat{\rho}) - \partial_t p_\Omega - \nabla p_\Omega \cdot \mathbf{w} - \nabla^2 p_\Omega + \nabla p_\Omega \cdot \nabla V_{ext} \\
&+ \int_\Omega (\nabla p_\Omega(r) + \nabla p_\Omega(r'))\rho(r')\nabla V_2(|r - r'|)dr' \\
&+ \int_{\partial\Omega} (p_{\partial\Omega}(r') - p_\Omega(r'))\rho(r')\frac{\partial V_2(|r - r'|)}{\partial n}dr' = 0.
\end{aligned} \tag{30}
$$

This is the adjoint equation with a non-local term. However, if the restriction on h is relaxed, such that $\frac{\partial h}{\partial n} \neq 0$ on $\partial\Omega$, then, additionally to the adjoint equation:

$$
\int_0^T \int_{\partial\Omega} \Big(p_{\partial\Omega} - p_\Omega\Big)\frac{\partial h}{\partial n}drdt = 0.
$$

This results in

$$
p_{\partial\Omega} - p_\Omega = 0, \tag{31}
$$

which can be compared to (21) in the Section 3.1.2. Using this relationship in the adjoint equation and setting $p := p_\Omega = p_{\partial\Omega}$, (30) reduces to:

$$
\begin{aligned}
&(\rho - \hat{\rho}) - \partial_t p - \nabla p \cdot \mathbf{w} - \nabla^2 p + \nabla p \cdot \nabla V_{ext} \\
&+ \int_\Omega (\nabla p(r) + \nabla p(r'))\rho(r')\nabla V_2(|r - r'|)dr' = 0.
\end{aligned} \tag{32}
$$

The next step is to relax the condition on h so that $h(T) \neq 0$, which recovers the final time condition on p:

$$
p(r, T) = 0,
$$

which can be compared with (19). Finally, the last restriction on h, $h = 0$ on $\partial\Omega$, can be removed, which results in:

$$
\begin{aligned}
&\int_0^T \int_{\partial\Omega} \Bigg(\frac{\partial p_\Omega}{\partial n} + p_\Omega \mathbf{w} \cdot \mathbf{n} - p_{\partial\Omega}\mathbf{w} \cdot \mathbf{n} + p_{\partial\Omega}\frac{\partial V_{ext}}{\partial n} - p_\Omega \frac{\partial V_{ext}}{\partial n} \\
&+ (p_{\partial\Omega} - p_\Omega) \int_\Omega \rho(r')\frac{\partial V_2(|r - r'|)}{\partial n}dr' \Bigg) h dr dt = 0,
\end{aligned}
$$

and applying relation (31) results in:

$$
\int_0^T \int_{\partial\Omega} \frac{\partial p}{\partial n} h dr dt = 0.
$$

Since this holds for all admissible h, the boundary condition for the adjoint equation is:

$$
\frac{\partial p}{\partial n} = 0, \quad \text{on} \quad \partial\Omega.
$$

This is equivalent to the boundary condition in Section 3.1.2, compare to (22). The full adjoint equation for the PDE-constrained optimization problem (27), applying (31) such that $p := p_{\partial\Omega} = p_\Omega$, is:

$$\begin{aligned}
&(\rho - \hat{\rho}) - \partial_t p - \nabla p \cdot \mathbf{w} - \nabla^2 p + \nabla p \cdot \nabla V_{ext} \\
&+ \int_\Omega (\nabla p(r) + \nabla p(r'))\rho(r')\nabla V_2(|r - r'|)dr' = 0, \quad \text{in} \quad \Omega, \\
&\frac{\partial p}{\partial n} = 0, \quad \text{on} \quad \partial\Omega, \\
&p(r, T) = 0.
\end{aligned}$$

3.2.2 The First Order Optimality System

Taking the Fréchet derivative of the Lagrangian (29) with respect to $\mathbf{w}$ is equivalent to the procedure in Section 3.1.3, since the non-local terms in the Lagrangian do not involve any terms that include $\mathbf{w}$. Moreover, by using the same argument as in Section 3.1.3, the state equation is recovered. The first-order optimality conditions for (27) are:

Adjoint Equation

$$\begin{aligned}
&(\rho - \hat{\rho}) - \partial_t p - \nabla p \cdot \mathbf{w} - \nabla^2 p + \nabla p \cdot \nabla V_{ext} \\
&+ \int_\Omega (\nabla p(r) + \nabla p(r'))\rho(r')\nabla V_2(|r - r'|)dr' = 0, \quad \text{in} \quad \Omega, \\
&\frac{\partial p}{\partial n} = 0, \quad \text{on} \quad \partial\Omega, \\
&p(r, T) = 0,
\end{aligned} \tag{33}$$

Gradient Equation

$$\beta\mathbf{w} - \rho\nabla p = 0 \quad \text{in} \quad \Omega,$$

Forward Problem

$$\begin{aligned}
&\partial_t \rho - \nabla^2 \rho + \nabla \cdot (\rho\mathbf{w}) - \nabla \cdot (\rho \nabla V_{ext}) \\
&-\nabla \cdot \int_\Omega \rho(r)\rho(r')\nabla V_2(|r - r'|)dr = 0, \quad \text{in} \quad \Omega, \\
&\frac{\partial \rho}{\partial n} - \rho\mathbf{w} \cdot \mathbf{n} + \rho\frac{\partial V_{ext}}{\partial n} + \int_\Omega \rho(r)\rho(r')\frac{\partial V_2(|r - r'|)}{\partial n}dr' = 0, \quad \text{on} \quad \partial\Omega, \\
&\rho(r, 0) = 0.
\end{aligned}$$

4 Numerical Methods

In order to solve the PDE-constrained optimization problem (27), it is necessary to solve the first-order optimality conditions (33). Therefore, methods of time and space discretization, as well as a method for solving the system of PDEs are needed. One challenge specific to this problem is the final time condition in the adjoint equation, which means that it is a boundary value problem in time as well as in space. The numerical methods that are needed to solve (33) are introduced in this section. A lot of research has been done on numerical methods for solving linear PDE-constrained optimization problems, as demonstrated in [12], [14] and [10]. New approaches to solving the optimality system (27) are needed because of the non-linear, non-local nature of the particle interaction term in the PDE-constraint. Standard methods are no longer sufficient to solve this type of problem, as discussed in this section.

4.1 Pseudospectral Methods

Pseudospectral methods on non-periodic domains are based on polynomial interpolation on non-equispaced points. Typically, Chebyshev points $\{x_j\}$ are chosen as collocation points on $[-1, 1]$, which are defined as:

$$x_j = \cos\left(\frac{j\pi}{N}\right), \quad j = 0, 1, ..., N, \tag{34}$$

see [15]. These points are clustered at the endpoints of the interval, and sparse around 0. Using this approach, the points are distributed from 1 to -1, which is counter-intuitive. Therefore, in the code library [8], which is used in producing the results of this report, the Chebyshev points are automatically flipped back to run from -1 to 1. Moreover, a linear map takes the points from the computational domain $[-1, 1]$ to any domain $[a, b]$ of interest. Interpolation on the Chebyshev grid is done using barycentric Lagrange interpolation, derived in [16]. The barycentric formula is:

$$p_N(x) = \frac{\displaystyle\sum_{k=0}^{N} \frac{\tilde{w}_k}{x - x_k} f(x_k)}{\displaystyle\sum_{k=0}^{N} \frac{\tilde{w}_k}{x - x_k}},$$

where the weights are defined as:

$$\tilde{w}_j = (-1)^j d_j, \quad d_j = \begin{cases} \frac{1}{2} \text{ for } j = 0,\ j = N, \\ 1 \text{ otherwise}, \end{cases}$$

see [16] and [8].

The derivation of the Chebyshev differentiation matrices is described below, following the presentation in [15]. The function of interest, f, is evaluated at the Chebyshev points $\{x_j\}$ and a grid function, $f_j := f(x_j)$, is defined. There exists a unique polynomial of degree $\leq N$ that can be used to interpolate f on the grid points x_j. The polynomial p satisfies, by definition, the following relationship:

$$p(x_j) = f_j, \tag{35}$$

so that the residual $p(x_j) - f_j$ is zero at these points. Therefore, this method is called a collocation method, see [17]. Once such a polynomial p is found, it can be differentiated and the following relationship is defined:

$$w_j = p'(x_j).$$

This relationship can be rewritten as a multiplication of f_j by a $(N+1) \times (N+1)$ matrix, denoted by D, as follows:

$$w_j = Df_j,$$

using (35). A $(N+1) \times (N+1)$ differentiation matrix D has the following entries, compare with [15]:

$$\begin{aligned}
(D)_{00} &= \frac{2N^2+1}{6}, \\
(D)_{NN} &= -\frac{2N^2+1}{6}, \\
(D)_{jj} &= -\frac{x_j}{2(1-x_j^2)}, \quad j = 1, ..., N-1, \\
(D)_{ij} &= \frac{c_i}{c_j}\frac{(-1)^{i+j}}{(x_i - x_j)}, \quad i \neq j, \quad i, j = 0, ..., N,
\end{aligned}$$

where

$$c_i = \begin{cases} 2, & i = 0 \text{ or } N, \\ 1, & \text{otherwise.} \end{cases}$$

The second derivative is represented by the second differentiation matrix D_2, which can be found by squaring the first differentiation matrix; $D_2 = D^2$, and more generally the j^{th} differentiation matrix is found as follows:

$$D_j = D^j.$$

However, in [8], the exact coefficients, derived in a similar way as above for D, are used to compute D_2, since it is more accurate than squaring D.

In order to extend the definition of the differentiation matrices to two-dimensional grids, a so-called tensor product grid has to be defined. First, Chebyshev points x_j, for $j = 1, ..., n$, on the x-axis and another set of Chebyshev points y_i, for $i = 1, ..., m$ on the y-axis are taken, both between $[-1, 1]$. Then the following two vectors are defined:

$$\mathbf{x}_K = (x_1, x_1, ..., x_1, x_2, x_2, ..., x_2, ..., x_n, x_n, ..., x_n)^T,$$
$$\mathbf{y}_K = (y_1, y_2, ..., y_m, y_1, y_2..., y_m, ..., y_1, y_2, ..., y_m)^T.$$

In $\mathbf{x}_K$, each x_j is repeated m times, while $\mathbf{y}_K$, each sequence $y_1, y_2, ..., y_m$ is repeated n times. The total length of each vector is $n \times m$. These vectors are defined, so that the set $(\mathbf{x}_K, \mathbf{y}_K)$ is a full set of all Chebyshev points on the two-dimensional tensor grid. Note that the points are clustered around the boundary of the two-dimensional grid and sparse in the middle of the grid. These Kronecker vectors can be used to find the Chebyshev differentiation matrices for two-dimensional problems as follows, compare to [15]. For a first derivative D in the x direction, a Kronecker product is taken of the one-dimensional Chebyshev differentiation matrix with the identity, as demonstrated here with three points:

$$D_x = I \otimes D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{pmatrix}$$
$$= \begin{pmatrix} d_{11} & d_{12} & d_{13} & & & & & & \\ d_{21} & d_{22} & d_{23} & & & & & & \\ d_{31} & d_{32} & d_{33} & & & & & & \\ & & & d_{11} & d_{12} & d_{13} & & & \\ & & & d_{21} & d_{22} & d_{23} & & & \\ & & & d_{31} & d_{32} & d_{33} & & & \\ & & & & & & d_{11} & d_{12} & d_{13} \\ & & & & & & d_{21} & d_{22} & d_{23} \\ & & & & & & d_{31} & d_{32} & d_{33} \end{pmatrix},$$

where the block structure matches the repetition of each x_j in $\mathbf{x}_K$. The second derivative with respect to x, D_{xx} can be found by using D_2 instead of D in this calculation. The derivative

with respect to y is found by taking the Kronecker product the other way around:

$$D_y = D \otimes I = \begin{pmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} d_{11} & & & d_{12} & & & d_{13} & & \\ & d_{11} & & & d_{12} & & & d_{13} & \\ & & d_{11} & & & d_{12} & & & d_{13} \\ d_{21} & & & d_{22} & & & d_{23} & & \\ & d_{21} & & & d_{22} & & & d_{23} & \\ & & d_{21} & & & d_{22} & & & d_{23} \\ d_{31} & & & d_{32} & & & d_{33} & & \\ & d_{31} & & & d_{32} & & & d_{33} & \\ & & d_{31} & & & d_{32} & & & d_{33} \end{pmatrix},$$

which now matches the repetition of each $y_1, ...y_m$ in $\mathbf{y}_K$. The Chebyshev differentiation of the Laplacian is given by: $L = I \otimes D_2 + D_2 \otimes I$.

In order to evaluate integrals in a similar way, the so-called Clenshaw–Curtis quadrature is used, which is derived in [18]. This is, for the integral over a smooth function f:

$$\int_{-1}^{1} f(x)dx = \sum_{k=0}^{N} w_k f(x_k), \tag{36}$$

where the weights are defined as:

$$w_j = \frac{2d_j}{N} \begin{cases} 1 - \sum_{k=1}^{(N-2)/2} \frac{2\cos(2kt_j)}{4k^2-1} - \frac{\cos(\pi j)}{N^2-1} & \text{for } N \text{ even}, \\ 1 - \sum_{k=1}^{(N-2)/2} \frac{2\cos(2kt_j)}{4k^2-1} & \text{for } N \text{ odd}, \end{cases}$$

see [8].
The advantage of Spectral Methods is that, for smooth functions, the convergence is exponential, see [17]:

$$\text{Pseudospectral Error} \approxeq O\left[\left(\frac{1}{N}\right)^N\right].$$

A good overview on spectral methods is given in [15] and a more in depth discussion can be found in [17].

4.2 Comparison with FEM and FDM

In this section the advantages and disadvantages of pseudospectral methods over finite element methods (FEM) and finite difference methods (FDM) are discussed, compare to [17]. The main difference between pseudospectral methods (PSM) and the other two methods is that PSM uses global basis functions on all Chebyshev points, while FEM uses local basis functions and FDM uses local, low order polynomials.

Finite difference methods use overlapping sequences of polynomials to approximate the solution of the problem. They are easy to implement and less costly per degree of freedom. However, they are also less accurate than PSM.

The basis functions used in FEM methods are generally of fixed, low degree and more accuracy is achieved through refinement of the elements; either in the whole domain or in regions where the problem is more difficult to solve. In comparison, the global basis function used in PSM are generally of higher degree than the ones in FEM. Furthermore, in order to refine the method, the degree of the polynomial can be increased.

In general, FEM results in large sparse matrix systems, which in many cases can be solved by exploiting their structure. It is also easily applied to complex domains, due to the shape of the elements. However, the disadvantage of FEM is low accuracy of solutions, due to the low degree of the polynomial basis functions. Furthermore, the sparsity property of the matrix systems is compromised when the PDEs involve nonlocal terms. Therefore, PSM are advantageous in this type of problems, since small dense matrices are utilized. As discussed in [19], an adaptive FEM method can be used to solve an integro-differential PDE-constrained optimal control problem. However, the accuracy achieved is mainly of order $O(10^{-2})$ and maximal $O(10^{-4})$, for a two dimensional problem with N nodes, where N is between $N = 3549$ and 50421. The time step is $dt = 0.05$. Furthermore, Dirichlet boundary conditions are used, which avoids applying more realistic no-flux boundary conditions. These no-flux boundary conditions are difficult to apply in the FEM context, because they are nonlocal as well. Within the existing code framework [8], these boundary conditions are straightforward to apply. The disadvantages of PSM are that they are more computationally expensive and the domain is required to be smooth. However, as discussed in [17], if the accuracy of PSM with $N = 10$ is to be achieved by FEM or FDM, a 10th order method has to be chosen with an error of $O(h^{10})$. All in all, PSM is the best method for solving PDE-contrained optimization problems involving integro-PDEs.

4.3 Exact Solutions

For some relatively simple PDE-constrained optimization problems it is possible to construct exact solutions to the first-order optimality system. This is an important aspect in the development of new numerical methods, since these problems can be used as test problems for

the numerical method and the exact error can be measured. The considered test problem is a simplified version of (11), and therefore testing the numerical method on this problem is a step towards solving (11). In this section, the construction of an exact solution for the following problem is derived, where the PDE constraint is the forced heat equation on $\Omega = [-1, 1]$:

$$\begin{aligned} &\min_{\rho,u} \quad \frac{1}{2}\|\rho - \hat{\rho}\|_{L_2}^2 + \frac{\beta}{2}\|u\|_{L_2}^2 \,, \\ &\text{subject to:} \\ &\partial_t \rho - \Delta\rho - u = 0, \quad \text{in} \quad \Omega, \\ &\rho(r,0) = \rho_0(r), \\ &\rho(r,t) = 0, \qquad\qquad \text{on} \quad \partial\Omega. \end{aligned} \tag{37}$$

Note that u is now the control variable, comparable to $\mathbf{w}$ in (11). The first-order optimality system for this PDE-constrained optimization problem is:

$$\begin{aligned} &\textbf{Adjoint Equation} \\ &\partial_t p + \Delta p - \rho + \hat{\rho} = 0 \qquad \text{in} \quad \Omega, \\ &p(r,T) = 0 \\ &p(r,t) = 0 \qquad \text{on} \quad \partial\Omega, \\ &\textbf{Gradient Equation} \\ &\beta u - p = 0 \qquad \text{in} \quad \Omega, \\ &\textbf{Forward Problem} \\ &\partial_t \rho - \Delta\rho - u = 0 \qquad \text{in} \quad \Omega, \\ &\rho(r,0) = \rho_0(r) \\ &\rho(r,t) = 0 \qquad \text{on} \quad \partial\Omega. \end{aligned} \tag{38}$$

This follows almost directly from taking the relevant terms from the optimality system (26). The solution to this system is derived in one and two dimensions, as well as for Dirichlet and Neumann boundary conditions.

In order to construct a full solution to the optimality system, the following steps have to be taken. At first, an expression for p has to be chosen, such that the boundary conditions for the adjoint equation, as well as the final-time condition, are satisfied. In a second step, this is substituted into the gradient equation to find u. The resulting expression can be used in the state equation to solve for ρ. Finally, once all three variables are defined, the adjoint equation can be used to solve for $\hat{\rho}$. A functional form for p, satisfying Dirichlet boundary conditions

and the final time condition is:

$$p(r,t) = \left(e^T - e^t\right)\cos(\pi r/2).$$

Substituting this into the gradient equation gives:

$$u(r,t) = \frac{1}{\beta}\left(e^T - e^t\right)\cos(\pi r/2).$$

Substituting u into the state equation results in a decoupled PDE for ρ that can be solved:

$$\partial_t \rho - \partial_r \rho = \frac{1}{\beta}\left(e^T - e^t\right)\cos(\pi r/2). \tag{39}$$

Assuming a solution of the form:

$$\rho(r,t) = \left(a + be^t\right)\cos(\pi r/2),$$

and substituting it into (39) gives:

$$be^t\cos(\pi r/2) + \frac{\pi^2}{4}\left(a + be^t\right)\cos(\pi r/2) = \frac{1}{\beta}\left(e^T - e^t\right)\cos(\pi r/2),$$

which results in:

$$\left(b + \frac{\pi^2}{4}b + \frac{1}{\beta}\right)e^t + \frac{\pi^2}{4}a - \frac{1}{\beta}e^T = 0.$$

Therefore, $b = -\dfrac{1}{(1+\frac{\pi^2}{4})\beta}$ and $a = \dfrac{4}{\beta\pi^2}e^T$, and so ρ becomes:

$$\rho(r,t) = \left(\frac{4}{\beta\pi^2}e^T - \frac{1}{(1+\frac{\pi^2}{4})\beta}e^t\right)\cos(\pi r/2).$$

The expressions for ρ and p can be substituted into the adjoint equation, to solve for $\hat{\rho}$:

$$e^t\cos(\pi r/2) + \frac{\pi^2}{4}\left(e^T - e^t\right)\cos(\pi r/2) = \hat{\rho} - \left(\left(\frac{4}{\beta\pi^2}e^T - \frac{1}{(1+\frac{\pi^2}{4})\beta}\right)\cos(\pi r/2)\right).$$

This gives:

$$\hat{\rho}(r,t) = \left(\left(\frac{4}{\beta\pi^2} + \frac{\pi^2}{4}\right)e^T + \left(1 - \frac{\pi^2}{4} - \frac{1}{(1+\frac{\pi^2}{4})\beta}\right)e^t\right)\cos(\pi r/2).$$

This solution can be used for Neumann boundary conditions as well, when considered on an interval $[-2,2]$. This is due to the fact that $\cos(\pi r/2)$ evaluated at $-2, 2$ is equal to -1 and 1 respectively, which is exactly at its stationary points. Therefore, the Neumann boundary conditions are satisfied at these points. Instead, the approach used in the numerical experiments

below is to slightly change the calculations above to derive the following exact solutions for ρ, p and $\hat{\rho}$ for solving (37) with Neumann boundary conditions:

$$
\begin{aligned}
p(r,t) &= \left(e^T - e^t\right)\cos(\pi r),\\
\rho(r,t) &= \left(\frac{1}{\pi^2\beta}e^T - \frac{1}{(1+\pi^2)\beta}e^t\right)\cos(\pi r),\\
\hat{\rho}(r,t) &= \left(\left(\pi^2 + \frac{1}{\pi^2\beta}\right)e^T + \left(1 - \pi^2 - \frac{1}{(1+\pi^2)\beta}\right)e^t\right)\cos(\pi r).
\end{aligned}
$$

Furthermore, these calculations can be done equivalently for two or more dimensional problems. The exact solution to the two-dimensional problem (37) with Dirichlet boundary conditions is:

$$
\begin{aligned}
p(r,t) &= \left(e^T - e^t\right)\cos(\pi x/2)\cos(\pi y/2),\\
\rho(r,t) &= \left(\frac{2}{\beta\pi^2}e^T - \frac{4}{(4+2\pi^2)\beta}e^t\right)\cos(\pi x/2)\cos(\pi y/2),\\
\hat{\rho}(r,t) &= \left(\left(\frac{2}{\beta\pi^2} + \frac{\pi^2}{2}\right)e^T + \left(1 - \frac{\pi^2}{2} - \frac{4}{(4+2\pi^2)\beta}\right)e^t\right)\cos(\pi x/2)\cos(\pi y/2),
\end{aligned}
$$

where $r = (x, y)$. Finally, the exact solution to the two-dimensional problem (37) with Neumann boundary conditions is:

$$
\begin{aligned}
p(r,t) &= \left(e^T - e^t\right)\cos(\pi x)\cos(\pi y),\\
\rho(r,t) &= \left(\frac{1}{2\beta\pi^2}e^T - \frac{1}{(1+2\pi^2)\beta}e^t\right)\cos(\pi x)\cos(\pi y),\\
\hat{\rho}(r,t) &= \left(\left(\frac{1}{\beta\pi^2} + 2\pi^2\right)e^T + \left(1 - 2\pi^2 - \frac{1}{(1+2\pi^2)\beta}\right)e^t\right)\cos(\pi x)\cos(\pi y).
\end{aligned}
$$

The exact solutions presented here for Dirichlet boundary conditions, for d dimensions, can be found in [20].

4.4 Multiple Shooting Method

Boundary value problem (BVP) solvers, such as `bvp4c`, are designed to treat BVPs in time, see [21]. Therefore, they are not equipped to deal with BVPs in both space and time. A method has to be developed that circumvents using BVP solvers and uses initial value problem (IVP) solvers instead. This strategy is called multiple shooting and the theoretical derivation of a multiple shooting approach for PDE-constrained optimization problems can be found in [22] and [14].

In this section, the numerical method is described at the different stages of its development. The PDE constraint considered has either Dirichlet or Neumann boundary conditions in space. The problem is treated in one and two dimensions. In order to initiate the development of

the method, a simpler PDE constrained optimization problem is considered at first. Once the method is established for the simpler problem, the non-local term is added. After that, two dimensional problems are considered.

4.4.1 One-Dimensional Diffusion

One of the simplest cases of a PDE-constrained optimization problem is heat control in one dimension. The PDE constraint involved is the forced heat equation, and the PDE-constrained optimisation problem (37), derived in Section 4.3, is used, and the derived optimality system is treated below.

The first step in solving this optimality system is to substitute the gradient equation into the heat equation for u and rearranging the equations to only have the time derivative on the left-hand side. This results in a coupled system of PDEs:

$$\begin{aligned}
&\partial_t \rho(r,t) = \partial_{rr}\rho(r,t) + \frac{1}{\beta}p(r,t), \qquad (40)\\
&\partial_t p(r,t) = -\partial_{rr}p(r,t) + \rho(r,t) - \hat{\rho}(r,t),\\
&\text{Initial and Final-Time Conditions:}\\
&\rho(r,0) = \rho_0(r),\\
&p(r,T) = 0,\\
&\text{Dirichlet Boundary Conditions:}\\
&\rho(r,t) = 0, \quad \text{on} \quad \partial\Omega,\\
&p(r,t) = 0, \quad \text{on} \quad \partial\Omega,
\end{aligned}$$

where $r \in [a,b]$ and $t \in [0,T]$. This system is considered as a test problem, since exact solutions for ρ and p are known. Therefore, the exact error of the numerical method can be determined at all points in space and time. The derivation of the exact solutions are discussed in Section 4.3.

The method that is used to solve this system of PDEs is called the shooting method. The procedure is to first discretize the problem in space, that is to replace the space derivatives with the appropriate Chebyshev differentiation matrices, as defined above. Then the problem reduces to a coupled system of ODEs, which can be solved using an ODE solver, such as the Matlab solver `ode15s`. The challenge is that the optimality system is a boundary value problem in time, since the adjoint equation has a final time condition in p. Therefore, the first idea is to create a guess for the initial condition $p_0(r)$, solve the coupled ODE system using this guess, extracting the computed p value at the final time, $p_{co}(T)$ and measuring the error between the computed p_{co} and the exact p_{ex}:

$$e = \left\| p_{co}(T) - p_{ex}(T) \right\|.$$

The final step in this procedure is to minimize this error by varying $p_0(r)$, using an in-built Matlab optimization routine, such as `fsolve`. This is easily implemented in Matlab, see Appendix A.1. However, the problem with this approach is that the adjoint equation, written in this form, is not well posed. The solution to this system blows up in finite time, which is caused by the negative diffusion term in the PDE for p.

Therefore, the adjoint equation has to be rewritten. This is done by rescaling time as

$$\tau = T + t_0 - t,$$

which causes the adjoint equation to run backwards in time from T to $t_0 = 0$. This changes the final time condition for p at time $t = T$, $p(T) = 0$, into an initial condition at time $\tau = 0$, $p(0) = 0$. The optimality system is then:

$$\begin{aligned}
&\partial_t \rho(r,t) = \partial_{rr}\rho(r,t) + \frac{1}{\beta}p(r,t), \qquad (41)\\
&\partial_t p(r,\tau) = \partial_{rr}p(r,\tau) - \rho(r,\tau) + \hat{\rho}(r,\tau),\\
&\text{Initial Conditions:}\\
&\rho(r,0) = \rho_0(r), \quad \text{for} \quad t = 0,\\
&p(r,0) = 0, \quad \text{for} \quad \tau = 0,\\
&\text{Dirichlet Boundary Conditions:}\\
&\rho(a,t) = \rho(b,t) = 0,\\
&p(a,\tau) = p(b,\tau) = 0,
\end{aligned}$$

where $t \in [t_0, T]$ and $\tau \in [T, t_0]$. This is now well posed. However, the issue with this rewritten system is that information about ρ at later times is needed to solve the adjoint equation and p values for earlier times are needed to solve the state equation, while neither of these information is available. Figure 1 visualises this problem. The initial conditions for ρ and p are represented as a green and blue dot respectively. Time t is represented by the green arrow, while time τ, the backwards time, is represented by a blue arrow. In order to test whether this approach works, the exact solution for ρ and q can be used, where information is missing. Then the problem is a decoupled system of PDEs, which is straightforward to solve.
In order to replace the missing information, as illustrated above, interpolation is used. Since interpolation using only the endpoints of the interval $[t_0, T]$ would be highly inaccurate, a strategy, called multiple shooting, is exploited in this section. The time interval is divided into a number of n subintervals, such that $t_0 < t_1 < t_2 < ... < t_n = T$. The subintervals are denoted by I_i, where $i = 0, 1, ..., n-1$. The values for ρ and p at these times constitute the initial guess. The discretization of the time interval and initial guesses for ρ and p are illustrated in Figure

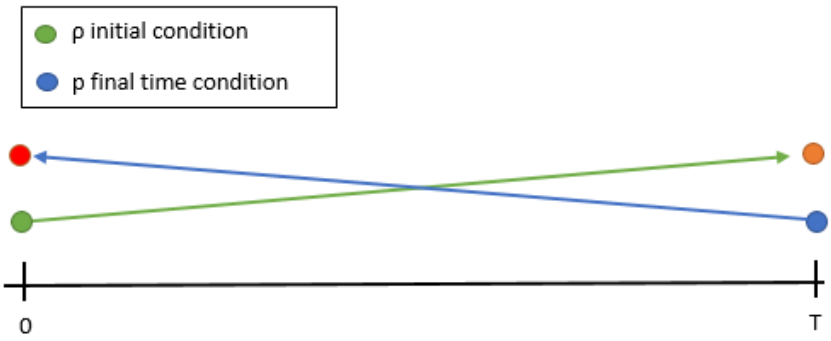

Figure 1: Solving a coupled system of PDEs on $[0, T]$.

2. The initial guess can be obtained by different methods, which will be discussed in a later section.

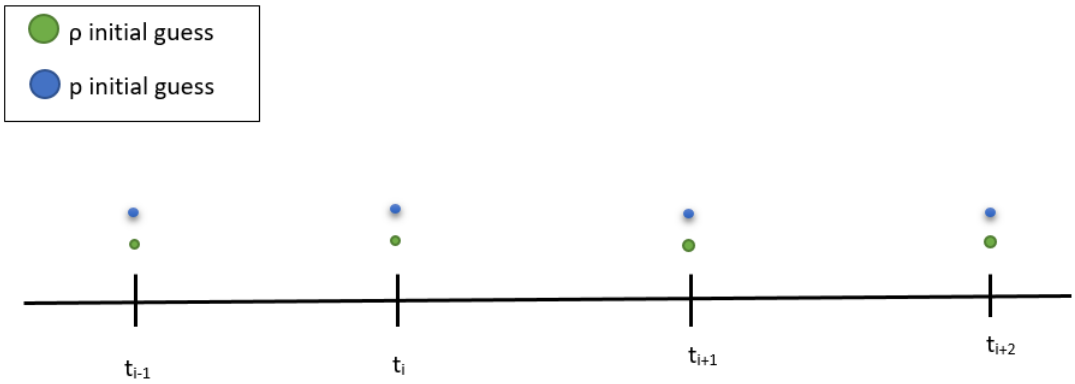

Figure 2: Discretizing the time interval and obtain initial guesses for ρ and q.

In a first step, these initial guesses are chosen to be the known exact solutions to ρ and p at the specified times t_i. The optimality system (41) is solved on each of the I_i, by considering the upper and lower bound of the subinterval, t_i and t_{i+1} instead of the global bounds t_0 and T. The new backward running time is defined, equivalently to above, as $\tilde{\tau} = t_{i+1} + t_i - t$, and the system becomes:

$$\partial_t \rho(r,t) = \partial_{rr}\rho(r,t) + \frac{1}{\beta}p(r,t), \tag{42}$$

$$\partial_t p(r,\tilde{\tau}) = \partial_{rr}p(r,\tilde{\tau}) - \rho(r,\tilde{\tau}) + \hat{\rho}(r,\tilde{\tau}),$$

Initial Conditions:

$$\rho(r,t_i) = \rho_{t_i}, \quad \text{for} \quad t = t_i,$$

$$p(r,t_{i+1}) = p_{t_{i+1}}, \quad \text{for} \quad \tilde{t} = t_{i+1},$$

Dirichlet Boundary Conditions:

$$\rho(a,t) = \rho(b,t) = 0,$$

$$p(a,\tilde{\tau}) = p(b,\tilde{\tau}) = 0,$$

where $t \in I_i = [t_i, t_{i+1}]$. On each subinterval, both ρ and p are interpolated between their known values at t_i and t_{i+1}, and the result is used to provide ρ at a later time step, to solve the adjoint equation, as well as p at an earlier time step, to solve the state equation.

In order to implement the strategy, (42) is evaluated on each time interval $I_i = [t_i, t_{i+1}]$, using interpolation for ρ in the adjoint equation and for p in the state equation to provide the missing information.

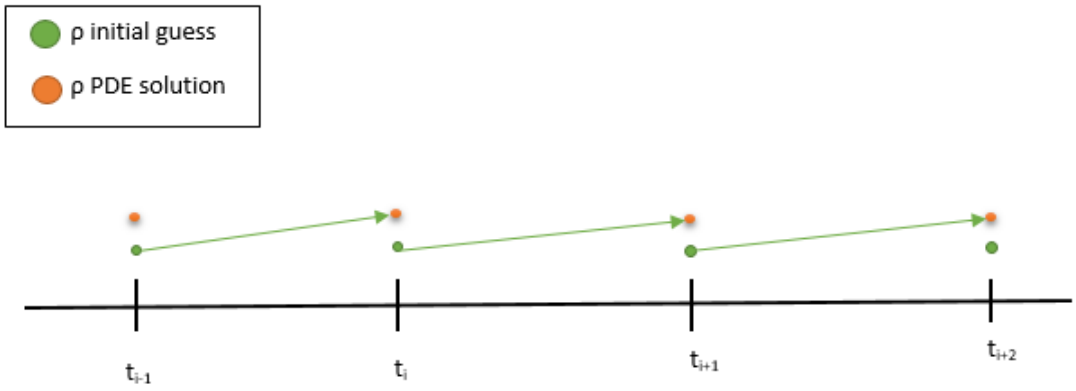

Figure 3: Solution strategy for ρ.

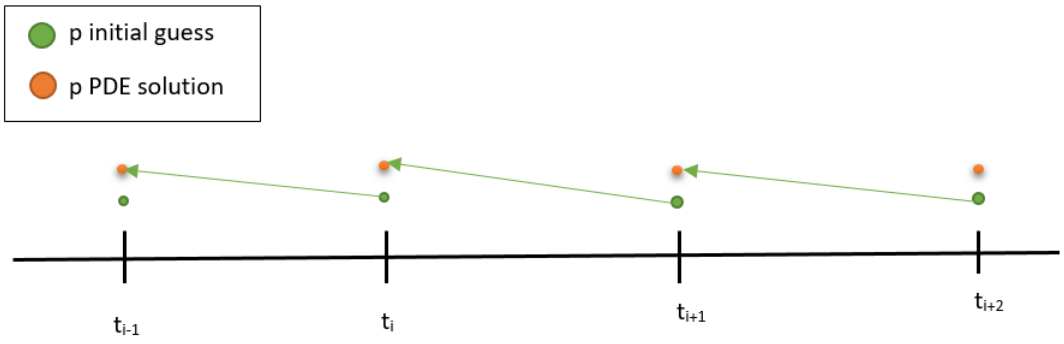

Figure 4: Solution strategy for p.

As can be observed in Figure 3, the value of ρ, taken from the ODE solver, at t_{i+1} is compared to ρ at t_{i+1}, which is the initial guess for solving (42) on the next interval $I_{i+1} = [t_{i+1}, t_{i+2}]$:

$$e = \|g_{init} - g_{sol}\|, \tag{43}$$

where $g_{init} = (g_1, g_2, ...g_n)$ is a vector of initial guesses for ρ on all n time points and g_{sol} is the vector of PDE solutions associated with the initial guesses on all time points. This

provides an error measure of the quality of the set of initial guesses g_{init} for ρ. This error calculation is repeated for p, as illustrated in Figure 4. However, since p is running backwards in time, the solution to the ODE solver provides the value for p at t_i, the lower bound on the considered interval I_i, which is then compared with the initial guess for p for the previous interval $I_{i-1} = [t_{i-1}, t_i]$. Note that for this solution strategy any ODE solver can be used to solve the discretized PDE on each subinterval. Furthermore, while pseudospectral methods are the best method for PDE-constrained optimization problems involving integro-PDE constraints, as discussed in Section 4.2, it is in principle possible to use other time or space discretization methods.

4.4.2 One-Dimensional Diffusion with a Non-Local Term

The problem (37) is extended by adding a non-local term to the forced heat equation. This non-local term is the two body interaction term that is introduced in Section 3.2. The one dimensional PDE-constrained optimization problem is:

$$\begin{aligned}
&\min_{\rho,u} \quad \frac{1}{2}\|\rho - \hat{\rho}\|_{L_2}^2 + \frac{\beta}{2}\|u\|_{L_2}^2\,, \\
&\text{subject to:} \\
&\partial_t \rho - \Delta\rho - u - \nabla \cdot \rho(r) \int_\Omega \nabla V_2(|r-r'|)\rho(r')dr' = 0, \quad \text{in} \quad \Omega, \\
&\rho(r,0) = \rho_0(r), \\
&\rho(r,t) = 0, \quad \text{on} \quad \partial\Omega.
\end{aligned} \tag{44}$$

The first-order optimality system, including the non-local term, is:

$$\begin{aligned}
&\textbf{Adjoint Equation} \\
&\partial_t p + \partial_{rr} p + \int_\Omega \Big(\partial_r p(r) + \partial_{r'} p(r')\Big)\rho(r')\partial_r V_2(|r-r'|)dr' = (\rho - \hat{\rho}) \quad \text{in} \quad \Omega, \\
&p(T) = 0 \\
&p(r,t) = 0 \qquad \text{on} \quad \partial\Omega, \\
&\textbf{Gradient Equation} \\
&\beta u - \rho = 0 \qquad \text{in} \quad \Omega, \\
&\textbf{Forward Problem} \\
&\partial_t \rho - \partial_{rr}\rho - u - \partial_r \rho(r) \int_\Omega \partial_r V_2(|r-r'|)\rho(r')dr' = 0 \qquad \text{in} \quad \Omega, \\
&\rho(r,0) = \rho_0(r), \\
&\rho(r,t) = 0 \qquad \text{on} \quad \partial\Omega.
\end{aligned} \tag{45}$$

This is directly derived from taking the relevant terms in (33). The system that has to be solved on each interval $[t_i, t_{i+1}]$ is, equivalent to (42):

$$\partial_t \rho(r,t) = \partial_{rr}\rho(r,t) + \frac{1}{\beta}p(r,t) + \partial_r \rho(r,t)\int_\Omega \partial_r V_2(|r-r'|)\rho(r',t)dr',$$
$$\partial_t p(r,\tilde{\tau}) = \partial_{rr}p(r,\tilde{\tau}) - \rho(r,\tilde{\tau}) + \hat{\rho}(r,\tilde{\tau}) - \int_\Omega \Big(\partial_r p(r,\tilde{\tau}) + \partial_{r'}p(r',\tilde{\tau})\Big)\rho(r',\tilde{\tau})\partial_r V_2(|r-r'|)dr',$$

Initial Conditions:

$$\rho(r,t_i) = \rho_{t_i}(r), \qquad \text{for} \quad t = t_i,$$
$$p(r,t_{i+1}) = p_{t_{i+1}}(r), \qquad \text{for} \quad t = t_{i+1},$$

Dirichlet Boundary Conditions:

$$\rho(a,t) = \rho(b,t) = 0,$$
$$p(a,\tilde{\tau}) = p(b,\tilde{\tau}) = 0,$$

where $\tilde{\tau} = t_{i+1} + t_i - t$, as before. As discussed above, V_2 is the force between two particles at positions r and r'. It is defined depending on the physical problem involved. For a first numerical test problem, the choice of V_2 is a Gaussian:

$$V_2(x) = \alpha e^{-x^2}. \tag{46}$$

Then $\partial_r V_2$ satisfies:

$$\partial_r V_2(|r-r'|) = -2\alpha|r-r'|e^{-|r-r'|^2}.$$

The specific problem that is solved numerically is:

$$\partial_t \rho(r,t) = \partial_{rr}\rho(r,t) + \frac{1}{\beta}p(r,t) + \alpha\partial_r \rho(r,t)\int_\Omega \partial_r e^{-|r-r'|^2}\rho(r',t)dr', \tag{47}$$
$$\partial_t p(r,\tilde{\tau}) = \partial_{rr}p(r,\tilde{\tau}) - \rho(r,\tilde{\tau}) + \hat{\rho}(r,\tilde{\tau}) - \alpha\int_\Omega \Big(\partial_r p(r,\tilde{\tau}) + \partial_{r'}p(r',\tilde{\tau})\Big)\rho(r',\tilde{\tau})\partial_r e^{-|r-r'|^2}dr',$$

Initial Conditions:

$$\rho(r,0) = \rho_0(r), \quad \text{for} \quad t = 0,$$
$$p(r,0) = 0, \qquad \text{for} \quad \tilde{\tau} = 0,$$

Dirichlet Boundary Conditions:

$$\rho(a,t) = \rho(b,t) = 0,$$
$$p(a,\tilde{\tau}) = p(b,\tilde{\tau}) = 0.$$

While the solution method is similar to the approach in Section 4.4.1, there are two key differences. Firstly, quadrature has to be used, employing (36), to evaluate the integral terms in the

optimality system. The other difficulty is that no exact solutions exist to this problem because of the complexity of the non-local term. Therefore, the main issue in solving (47) is finding an initial guess on the Chebyshev time points t_i, which is close enough to the solution of the system, so that convergence to a continuous solution on the whole interval is possible. The initial guess is found by using a technique called continuation.

The parameter α in (47) represents the strength of the contribution of the integral term to the system of PDEs. It can be varied to choose the strength of the particle interactions on the PDE solution. If $\alpha_0 = 0$, there is no particle interaction and the optimality system for the forced heat equation is recovered, compare to (42). Since the solution to that problem is known, the idea is to use the exact solution to the problem where $\alpha_0 = 0$ as an initial guess to the problem where α_1 is non-zero but small. The optimal initial guess for the problem involving α_1 is found by multiple shooting and used as an initial guess for the problem with α_2, where $\alpha_2 > \alpha_1$. This process is repeated iteratively until a certain contribution of the integral term is reached, for example $\alpha = 1$.

Generally, the step size in α is not determined linearly, but chosen adaptively. This is because some parts of the problems may be easier to solve than others. If the change in α is chosen small, then the optimization function only needs a few iteration to find the optimal initial guess, based on the result of the previous step in α. This is because the problem with α_{i+1} is similar to the problem involving α_i and therefore the optimal initial guesses for the two problems are close. The downside of this approach is that the problem has to be re-evaluated for many different values of α, which is computationally expensive. If α_{i+1} is chosen to be considerably larger than α_i at each step, the problem has to be solved less times. However, the risk is that the optimal initial guess for the problem with α_i is not a suitable initial guess for the problem with α_{i+1}, and that no solution is found. There are many ways to change α adaptively and it depends greatly on the problem that is to be solved.

4.4.3 Two-Dimensional Problems

The two dimensional version of the PDE constrained optimization problem (37) is treated with numerical methods equivalent to the ones used in one-dimensional case, as discussed in Section 4.4.1. The corresponding optimality system is (38). All the solution methods follow directly from the one-dimensional method presented in Section 4.4.1. The only difference is that, instead of having a one-dimensional set of spatial Chebyshev points, a two-dimensional Chebyshev grid has to be used, making use of Kronecker products. This has been introduced in Section 4.1. One of the things to note is that the computational effort in two dimensions is much higher than for one dimensional calculations. This is because instead of N spatial points, $N_1 \times N_2$ spatial points have to be evaluated.

4.4.4 Two-Dimensional Problems with a Non-Local Term

Adding a non-local term to (37), the PDE-constrained optimization problem becomes:

$$\min_{\rho,u} \quad \frac{1}{2}\|\rho-\hat{\rho}\|_{L_2}^2+\frac{\beta}{2}\|u\|_{L_2}^2\,,$$

subject to:

$$\begin{aligned}
&\partial_t\rho=\Delta\rho+u+\nabla\cdot\int_\Omega\rho(r)\rho(r')\nabla V_2(|r-r'|)dr' \quad \text{in} \quad \Omega,\\
&\rho(r,0)=\rho_0(r),\\
&\rho(r,t)=0, \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{on} \quad \partial\Omega,
\end{aligned}$$

where $r=(x,y)\in\mathbf{R}^2$. The corresponding optimality system is:

Adjoint Equation

$$\begin{aligned}
\partial_t p(r,\tau)&=\Delta p(r,\tau)-\rho(r,\tau)+\hat{\rho}(r,\tau)\\
&\quad-\int_\Omega\Big(\nabla_r p(r,\tau)+\nabla_{r'}p(r',\tau)\Big)\rho(r',t)\nabla_r V_2(|r-r'|)dr' \qquad \text{in} \quad \Omega,\\
p(T)&=0\\
p(r,t)&=0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{on} \quad \partial\Omega,
\end{aligned}$$

Gradient Equation

$$\beta u-\rho=0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{in} \quad \Omega,$$

Forward Problem

$$\begin{aligned}
\partial_t\rho(r,t)&=\Delta\rho(r,t)+u(r,t)+\nabla_r\cdot\int_\Omega\rho(r,t)\rho(r',t)\nabla_r V_2(|r-r'|)dr' \quad \text{in} \quad \Omega,\\
\rho(r,0)&=\rho_0(r),\\
\rho(r,t)&=0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{on} \quad \partial\Omega,
\end{aligned}$$

compare with the one-dimensional system (45). Equivalently to the one-dimensional particle interaction term, (46), V_2 is the two-dimensional Gaussian:

$$V_2(x,y)=\alpha e^{-(x^2+y^2)}.$$

When substituting the gradient equation into the state equation, the optimality system becomes:

$$\partial_t \rho(r,t) = \Delta\rho(r,t) + \frac{1}{\beta}p(r,t) + \alpha\nabla_r \cdot \int_\Omega \nabla_r\left(e^{-|r-r'|^2}\right)\rho(r',t)\rho(r,t)dr', \tag{48}$$

$$\partial_t p(r,\tilde{\tau}) = \Delta p(r,\tilde{\tau}) - \rho(r,\tilde{\tau}) + \hat{\rho}(r,\tilde{\tau}) - \alpha\int_\Omega \left(\nabla_r p(r,\tilde{\tau}) + \nabla_{r'} p(r',\tilde{\tau})\right)\cdot\nabla_r\left(e^{-|r-r'|^2}\right)\rho(r',\tilde{\tau})dr',$$

Initial Conditions:

$$\rho(r,0) = \rho_0(r), \quad \text{for} \quad t = 0,$$

$$p(r,0) = 0, \quad \text{for} \quad \tilde{\tau} = 0,$$

Dirichlet Boundary Conditions:

$$\rho(a,t) = \rho(b,t) = 0,$$

$$p(a,\tilde{\tau}) = p(b,\tilde{\tau}) = 0,$$

where $\tilde{\tau} = T + t_0 - t$. The method for solving this system, including multiple shooting and continuation, follows exactly from the one-dimensional approach discussed in Section 4.4.2.

5 Numerical Experiments

All numerical methods described in this section are implemented in Matlab. The pre-existing code library in [8] provides an ideal starting point, since it is written to solve the multiscale particle dynamics PDEs, such as (1), using pseudospectral methods. It also contains a straightforward method of applying the spatial boundary conditions. In all experiments $\beta = 10^{-3}$ is chosen and the computational domain is $[-1,1]$ in each spatial direction and $[0,1]$ in time. All error calculations consider the relative error using the L^2 norm in the space variable and the L^∞ norm in time:

$$\left\| \|y_{num} - y_{ex}\|_{L^2(\Omega)} \right\|_{L^\infty([0,1])} = \max\left[\left(\frac{\int_\Omega (y_{num} - y_{ex})^2 dx}{\int_\Omega y_{ex}^2 dx}\right)^{1/2}\right],$$

where y represents the variable of interest, y_{ex} is the exact value and y_{num} is the numerical value of y. The error is scaled by the square root of the number of space points N to allow for comparison between computations with different number of points.

5.1 One-Dimensional Diffusion

In this section the solutions to the problems presented in Section (4.4.1) are numerically implemented and some results are presented. The strategy developed in the previous section is first tested using exact solutions instead of interpolated ρ and q values on each interval. This

is solving (42), substituting ρ_{Ex} and p_{Ex} as follows:

$$\partial_t \rho(r,t) = \partial_{rr}\rho(r,t) + \frac{1}{\beta} p_{Ex}(r,t),$$
$$\partial_t p(r,\tilde{\tau}) = \partial_{rr} p(r,\tilde{\tau}) - \rho_{Ex}(r,\tilde{\tau}) + \hat{\rho}(r,\tilde{\tau}).$$

Running the Matlab code shows that the solution from the ODE solver agrees with the exact solution to the order of 10^{-11} most of the time. In Table 1 the errors in the ODE solver for different number of space points N and time points n can be observed for Dirichlet boundary conditions, and in Table 2, the same data is shown for Neumann boundary conditions. The ODE tolerances for these computations are of order 10^{-12}.

N	$n = 11$	$n = 21$	$n = 41$
10	7.5743×10^{-9}	6.9957×10^{-9}	6.5771×10^{-9}
20	2.1284×10^{-11}	7.0487×10^{-11}	1.7138×10^{-10}
30	1.1025×10^{-11}	4.2646×10^{-11}	1.5328×10^{-10}
40	1.0260×10^{-11}	3.0101×10^{-11}	1.4187×10^{-10}
100	1.0270×10^{-11}	1.2649×10^{-11}	6.8519×10^{-11}

Table 1: Error between ODE solver and exact solution, Dirichlet Boundary Conditions.

N	$n = 11$	$n = 21$	$n = 41$
10	1.2871×10^{-4}	7.2612×10^{-5}	4.2279×10^{-5}
20	5.4343×10^{-12}	6.2952×10^{-12}	6.7937×10^{-12}
30	3.4268×10^{-12}	2.9423×10^{-12}	3.9497×10^{-12}
40	4.4606×10^{-12}	5.5659×10^{-12}	6.5336×10^{-12}
100	2.8356×10^{-12}	6.1763×10^{-12}	1.8959×10^{-11}

Table 2: Error between ODE solver and exact solution, Neumann Boundary Conditions.

Since this shows that the ODE solver provides good solutions, given the exact solutions instead of coupling the two ODEs, this method can be investigated further. The code can be found in Appendix A.2.

Next, the exact solution in the two ODEs is replaced by an interpolated solution, using the exact values of ρ and $\tilde{p}$ as initial guesses at the t_i, as illustrated in Figure 2. The time discretization points t_i are changed to be Chebyshev points, and the initial guess for ρ and p on these t_i are computed, using the exact solution. The aim is to find ρ and p on a finer, equispaced grid, using the interpolation matrix, which is part of the code library in [8]. The finer grid is discretized into $m > n$ equispaced time points $\tilde{t}_j$, where $j = 0, 1, ..., m$. The resulting interpolation matrix can be used and the interpolated ρ_{Int} and p_{Int} can then be substituted into the ODE to replace the missing information for ρ at the later time steps and for p at earlier times. Again, (42) is

solved, substituting ρ_{Int} and p_{Int} as follows:

$$\partial_t \rho(r,t) = \partial_{rr}\rho(r,t) + \frac{1}{\beta} p_{Int}(r,t),$$
$$\partial_t p(r,\tilde{\tau}) = \partial_{rr} p(r,\tilde{\tau}) - \rho_{Int}(r,\tilde{\tau}) + \hat{\rho}(r,\tilde{\tau}).$$

Time and interpolation points: $n = 20, nInt = 1000$			
N	ρ	p	ODE solver
10	4.3120×10^{-16}	4.2381×10^{-14}	1.6341×10^{-6}
20	3.9323×10^{-16}	3.5660×10^{-14}	1.3739×10^{-6}
30	3.6160×10^{-16}	3.2268×10^{-14}	1.2417×10^{-6}
40	3.0266×10^{-16}	3.0018×10^{-14}	1.1555×10^{-6}
100	2.4226×10^{-16}	2.3836×10^{-14}	9.1888×10^{-7}

Table 3: Error between interpolated and exact values in ρ, p and the ODE solution, Dirichlet Boundary Conditions.

Time and interpolation points: $n = 20, nInt = 1000$			
N	ρ	p	ODE solver
10	8.3538×10^{-11}	4.2523×10^{-14}	8.6811×10^{-6}
20	7.2130×10^{-11}	3.5652×10^{-14}	7.3000×10^{-6}
30	6.3615×10^{-11}	3.2161×10^{-14}	6.5962×10^{-6}
40	6.0907×10^{-11}	3.0020×10^{-14}	6.1385×10^{-6}
100	4.6959×10^{-11}	2.3944×10^{-14}	4.8819×10^{-6}

Table 4: Error between interpolated and exact values in ρ, p and the ODE solution, Neumann Boundary Conditions.

The relative error between the exact solutions of ρ and p and their interpolated values at $\tilde{t}_j$, the finer grid points, is of order 10^{-16} and 10^{-14}, respectively. However, the error in the ODE solution, when using the interpolated values for ρ and p, is of order 10^{-6}, even though the ODE solver tolerance is set to be of order 10^{-12}. This can be seen for both Dirichlet and Neumann spacial boundary conditions in Table 3 and Table 4, respectively, for number time points $n = 20$ and interpolation points $nInt = 1000$. Note that varying n and $nInt$ yields similar results, which are omitted in this report for brevity. This mismatch in errors is caused by the fact that the ODE solver evaluates ρ and p at times t other than $\tilde{t}_j$. However, the way the Matlab code is written, the solver is forced to choose the value of ρ and $\tilde{p}$ at the closest $\tilde{t}_j$ to the time t, which causes the heightened error. The Matlab code for this part of the strategy,

for one time interval $[t_i, t_{i+1}]$, can be found in Appendix A.3.

Since the ODE solver requires times t that are not part of the interpolating time grid $\{\tilde{t}_j\}$, the following approach is taken. The interpolation is not done on a specified time grid $\{\tilde{t}_j\}$, but at each time t that is requested by the ODE solver. This is done using Chebyshev interpolation with one time point at a time, which is part of the code library [8]. The resulting ρ_{Int} and p_{Int} can be substituted into the ODEs to replace the missing information, as before. This is computationally more expensive, but provides more accurate solutions. Again, the error between the ODE solution, given the exact values for ρ and p, and the ODE solution, given interpolated ρ and p, is computed. Here the ODE solver tolerances are adjusted to be of order 10^{-9}, since, when applying Neumann boundary conditions, the problem becomes too stiff to solve with higher ODE solver tolerances. The resulting error is of order 10^{-12} to 10^{-15}, which is a great improvement compared to the error of the ODE solution when using the equispaced interpolation grid, which was of order 10^{-6}. The comparison of the error in the ODE solution using the equispaced grid and pointwise Chebyshev interpolation can be found in Table 5 for Dirichlet boundary conditions and in Table 6 for Neumann boundary conditions. The Matlab code, for one time interval $[t_i, t_{i+1}]$, can be found in Appendix A.4. This code can then be extended to compute the consecutive intervals, I_i, for all $i = 1, 2, ..., n$. Note that this process can be parallelized by computing the n time intervals at the same time. This could make the computations up to n times faster, depending on the machine used to run the code on.

	$n = 20$		$n = 50$	
N	Equispaced ($nInt = 1000$)	Chebyshev (pw)	Equispaced ($nInt = 1000$)	Chebyshev (pw)
10	1.8946×10^{-6}	1.6327×10^{-15}	1.8944×10^{-6}	2.8408×10^{-15}
20	1.8212×10^{-6}	2.9466×10^{-15}	1.8213×10^{-6}	3.9014×10^{-15}
30	1.5926×10^{-6}	3.0336×10^{-15}	1.5927×10^{-6}	2.7717×10^{-15}
40	1.5390×10^{-6}	5.1776×10^{-15}	1.5390×10^{-6}	4.5207×10^{-15}
100	1.2382×10^{-6}	8.2713×10^{-15}	1.2384×10^{-6}	1.1878×10^{-14}

Table 5: Error in the ODE solution using an equispaced grid and pointwise (pw) Chebyshev interpolation, Dirichlet Boundary Conditions.

Since this approach works well when the initial guesses are exact solutions, in a final step, the exact solution on the t_i is perturbed by a small amount. This perturbed initial guess is passed to an optimization routine in Matlab, `fsolve`, which minimizes the error, (43), between the initial guesses and the resulting ODE solutions at each t_i, by varying the set of initial guesses. The optimized set of initial guesses should be close to the exact solution on the t_i. As an

	$n = 20$		$n = 50$	
N	Equispaced ($nInt = 1000$)	Chebyshev (pw)	Equispaced ($nInt = 1000$)	Chebyshev (pw)
10	8.3538×10^{-6}	3.6301×10^{-13}	9.1304×10^{-6}	4.9008×10^{-13}
20	7.6878×10^{-6}	1.8867×10^{-13}	7.6872×10^{-6}	4.5723×10^{-13}
30	6.9087×10^{-6}	4.9385×10^{-15}	6.9577×10^{-6}	3.1387×10^{-14}
40	6.5328×10^{-6}	5.8337×10^{-15}	6.5222×10^{-6}	1.7197×10^{-14}
100	5.2001×10^{-6}	6.5326×10^{-14}	5.2081×10^{-6}	3.5702×10^{-14}

Table 6: Error in the ODE solution using an equispaced grid and pointwise (pw) Chebyshev interpolation, Neumann Boundary Conditions.

initial test, the optimization routine is given the exact initial guess. The optimization routine should take this initial guess and find that it is the correct solution within one iteration. This is demonstrated in Appendix A.5.

Then, the code in Appendix A.5 is extended by wrapping a data storage function around it. This is to save the computed solutions for each perturbation of the initial guess externally in a MAT file. The data storage code furthermore sets up a loop, which performs the optimization discussed above and stores the data for a list of perturbations of the initial guess and parameters, which can be changed for different computations. The perturbations are achieved using two perturbation functions, one that adds a deterministic, the other a random perturbation to different locations in space and time of the exact initial guess vector. The functions take the locations and strength of the perturbation as input and return a perturbed initial guess, which is fed into the optimization routine. The code for this can be found in Appendix A.6. Results produced can be seen in Table 8 for Dirichlet boundary conditions and Table 9 for Neumann boundary conditions, which can both be found in Appendix C.1. The tables show the perturbation locations and strengths for both deterministic and random perturbations. Note that the initial and final times as well as the spatial boundaries are not perturbed, since otherwise the problem that is solved would change. The `fsolve` error and the number of function evaluations are displayed. The `fsolve` tolerances are of order 10^{-6}, while the ODE solver tolerances are of order 10^{-8}. It is important to have at least one order difference between these tolerances, because if the accuracy of the ODE solution is similar to the accuracy of the optimization, the optimization becomes meaningless. The results for the problems with Dirichlet and Neumann boundary conditions are very accurate with an error of orders between 10^{-5} and 10^{-7} mostly. There are three points in Table 9, where no solution could be found. When changing the number of spatial points from $N = 20$ to $N = 30$ and the ODE tolerance from 10^{-8} to 10^{-9} in order to increase the difference to the `fsolve` tolerance, the problems were

solved. Their error is included in the table in brackets behind the original solution.

5.2 One-Dimensional Diffusion with a Non-Local Term

In order to solve the optimality system (47), involving the non-local term, derived in Section 4.4.2, the same approach is taken as in Section 5.1. The continuation method employed in this report is to increase α at some small rate until a threshold in a predefined quantity is reached. This threshold is chosen to be a certain error at the t_i between the solution to the problem and the initial guess. However, in gerneral this could be replaced by another measure of the difficulty of the problem, such as number of iterations of `fsolve` or computational time. After the threshold is reached, the problem is evaluated at the corresponding α_i, using multiple shooting. The resulting new optimal initial guess is used to evaluate the problem at α_{i+1}. The value of α_{i+1} is found again by increasing the value from α_i until the threshold is met. This is only one simple example of changing α. The choice of α is highly problem dependent and there are many ways to choosing the relevant α_i's. In Table 10, the results for the continuation code in Appendix A.7 are displayed for increasing α up to $\alpha \approxeq 0.1$, when using $N = 20$ and $N = 40$ spatial points. The table can be found in Appendix C.2. The intermediate values of α are chosen as follows. Starting from $\alpha = 0.0001$, its value is increased by multiplying by 0.1 until the threshold 10^3 in the error function is reached. The error function can also be found in Appendix A.7. Then the `fsolve` routine is evaluated and an optimal initial guess for the problem with the α corresponding to the threshold is found. This optimal initial guess is used as the initial guess for the problem using a larger α, found again by evaluating the error function until reaching the threshold value. After $\alpha = 0.05$, α is multiplied by 0.01 instead until the new threshold 10^2 is reached. After $\alpha = 0.065$, the multiplicative factor is 0.001 and the error threshold is 50. It is assumed that the problem becomes more difficult the larger α is, and therefore smaller steps should be taken as α increases. Note that the code is computing the problem with Dirichlet boundary conditions on $N = 20$ points in space, on the interval $[-1, 1]$ and $n = 11$ points in time. The tolerance for the first order optimality in the `fsolve` function is 0.1.

The solution to the optimality system can be plotted for the different values of α. In Figure 5 the results for ρ and p for the problem with $\alpha = 0$ and $\alpha = 0.1$ can be compared, for $N = 20$ and $n = 11$. Each line correspond to one of the t_i Chebyshev time points. Note that the choice of positive α implies that the particles are repulsive, which is exactly what can be observed in Figure 5 for $\alpha = 0.1$. Choosing a negative α instead would represent attractive particles. However, this case is omitted in this report, since it would require changing the target $\hat{\rho}$ in the test problem. Evaluating the current problem with a negative α, without changing $\hat{\rho}$, would cause the attractive particles to work against reaching the target. Therefore, either the target would not be reached or β would have to be smaller to allow for more energy to be spent on

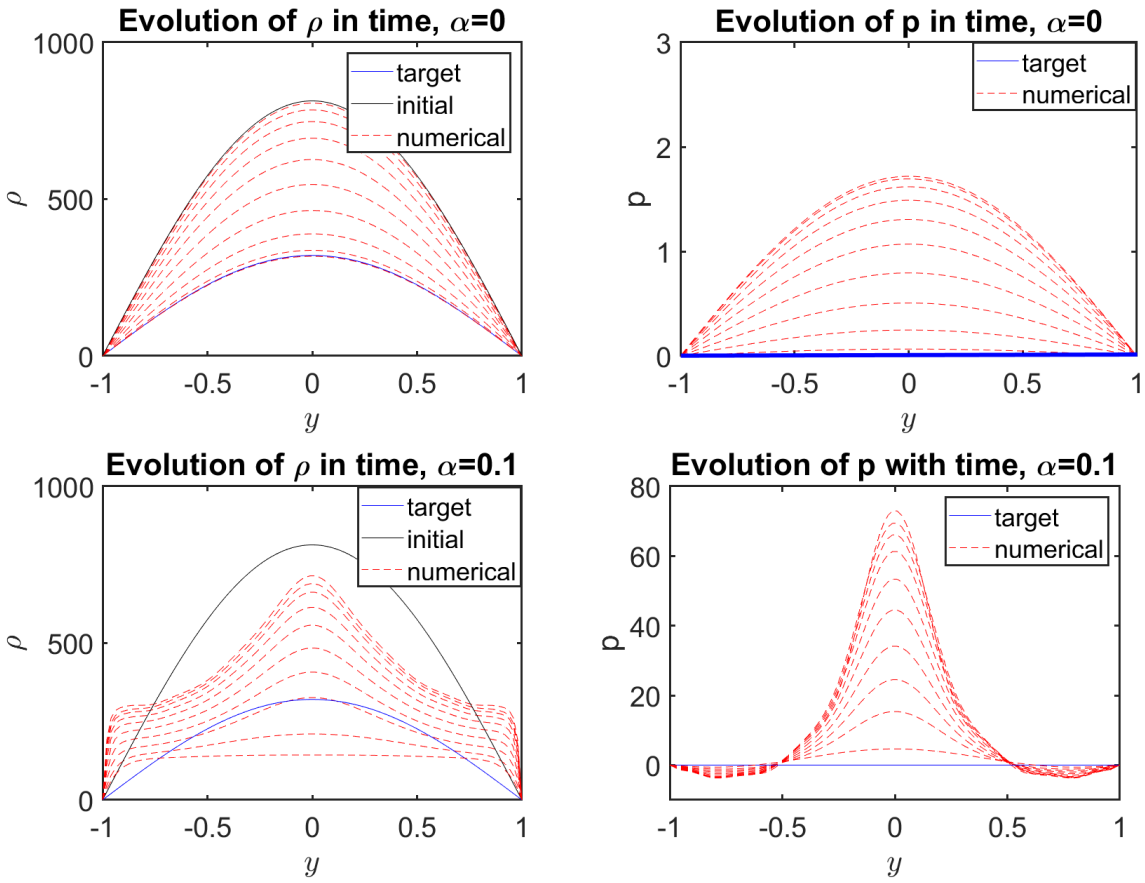

Figure 5: Solution to the optimality system with $\alpha = 0$ (top) and $\alpha = 0.1$ (bottom), where $N = 20$, $n = 11$.

reaching $\hat{\rho}$.

In Figure 6, solutions to the optimality system for different numbers of space points N, and $n = 11$ time points are plotted. The value of α is similar for each N: For $N = 10$, $\alpha = 0.0672$, for $N = 20$, $\alpha = 0.0692$, for $N = 30$, $\alpha = 0.0685$ and for $N = 40$, $\alpha = 0.0685$. The values for α are not completely identical due to the continuation procedure that determines the α values, as described above. Note that for $N = 10$, $\alpha = 0.0672$ is the highest value of α that can be achieved. For larger α, the method does not converge. This suggests that $N = 10$ are not enough points to achieve numerical stability. Figure 6 shows that, while the solutions for $N = 20$, 30 and 40 are almost identical, the solution for $N = 10$ is significantly different. This is another reason why $N = 10$ are not enough points to get a numerically stable solution. However, if enough points are added, the solutions converge for different choices of space points at a similar choice of α. This supports that the numerical method developed in this report for solving optimality systems involving integro-PDEs is stable and converges. This method has not been tested for other boundary conditions yet, because it requires changing the test problem that is solved. In this framework, solving the problem with no-flux boundary conditions is not physically meaningful.

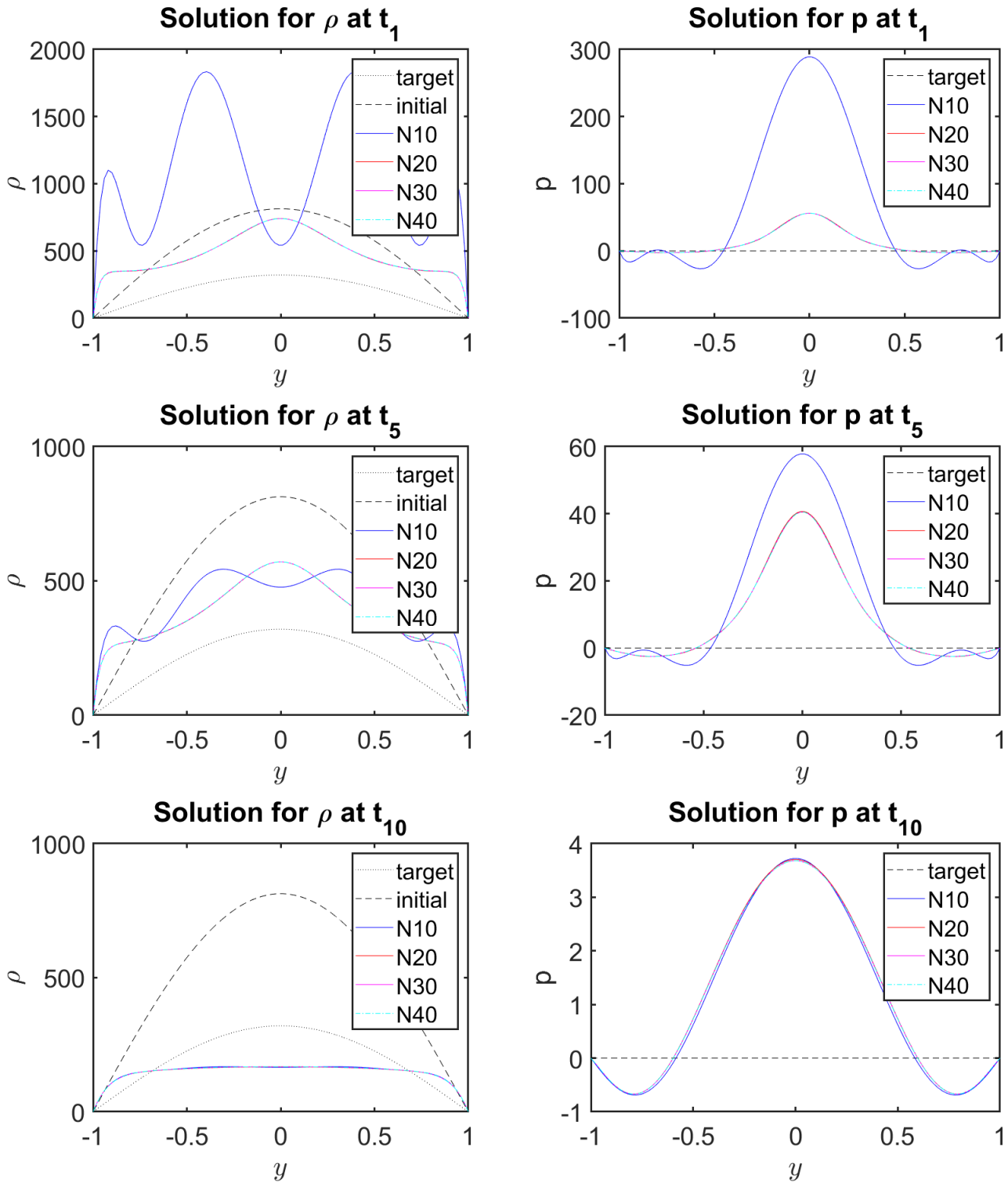

Figure 6: Solution to the optimality system with $\alpha \approxeq 0.068$ for $N = 10, 20, 30, 40$ and $n = 11$.

5.3 The Two-Dimensional Problem

Solving the two-dimensional problem (37) is almost equivalent to solving the one-dimensional problem, as described in the previous section. The few changes in setting up the problem, as discussed in Section 4.4.3, can be seen in Appendix B.1. The `fsolve` tolerances are set to be of order 10^{-6}, while the ODE solver tolerances are of order 10^{-8}. The optimality tolerance is 0.1. The numerical experiments in this sections are done with $N_1 = 10$ and $N_2 = 10$, since the overall number of points is then $N_1 \times N_2 = 100$, which is computationally expensive in comparison to $N = 50$ in one dimension. The exact solutions used to provide an initial guess, as in the one dimensional case, as taken from Section 4.3.

In this report, only the Dirichlet problem is considered in two dimensions. The Neumann problem is computationally more expensive and takes more time to compute. It is however included in the code in Appendix B.1. As in the one-dimensional case, the optimisation routine is tested by giving it the exact solution to the two-dimensional problem as an initial guess. As before, this optimization is done in one iteration. Since this shows that the code works for two-dimensional problems as well, the initial guess is perturbed away from the exact solution, and the error is computed. This is equivalent to the procedure in one dimension, however, since the computations take much longer than in one dimension, only a few perturbations are tested to illustrate the method. The results are presented in Table 7. These results can be compared to the ones in Table 8 for the one dimensional case, showing that similar orders of accuracy are achieved in the one- and two-dimensional calculations.

Perturbation Location			Perturbation		Deterministic Pert.		Random Pert.	
ρ	q	t	ρ	q	Error	F.Eval.	Error	F.Eval.
none	none	none	0	0	0	2001	0	2001
1(m)	none	4	0.01	0	1.8059×10^{-5}	2001	3.6921×10^{-6}	2001
1(m)	none	1 (e)	0.1	0	1.5150×10^{-7}	4002	1.4340×10^{-7}	4002
1(m)	none	1 (m)	1	0	1.8293×10^{-7}	4002	1.5393×10^{-7}	4002
all	none	all	0.01	0	3.5571×10^{-7}	4002	2.7070×10^{-7}	4002
1(m)	1(m)	all	0.01	0.01	3.7356×10^{-7}	4002	1.5030×10^{-7}	4002
1(m)	1(m)	all	0.1	0.1	6.7863×10^{-7}	4002	1.0902×10^{-6}	4002
1(m)	1(m)	all	1	1	1.4699×10^{-6}	8004	4.1318×10^{-7}	6003
all	all	all	0.1	0.01	1.1586×10^{-6}	6003	1.0594×10^{-6}	6003

Table 7: Perturbation locations (m=middle, e=end), Perturbation strength (deterministic/random), final error in ODE solver after fsolve, number of function evaluations (F.Eval.), Dirichlet BCs., 2D

5.4 The Two-Dimensional Problem with a Non-Local Term

Solving the optimality conditions (48) for the two-dimensional problem, involving the non-local term, follows almost exactly from the one-dimensional computations. The continuation method and determining the different values of α remains unchanged and is detailed in Section 5.2. The code for the two-dimensional calculations only slightly differs due to the set up of the two dimensional Gaussian, the two dimensional grid and differentiation matrices and consequently the implementation of the integral term. The changes in the code can be found in Appendix B.2. As in the previous section, `fsolve` tolerances are of order 10^{-6} and ODE solver tolerances are of order 10^{-8}. The optimality tolerance is 0.1 and the number of spatial points on each axis are $N_1 = 10$ and $N_2 = 10$. Some results of the continuation method in two dimensions can be found in Table 11 in Appendix C.3, which can be compared to the one dimensional results in Table 10. Note that α is not increased up to $\alpha = 0.1$, as done in the one-dimensional case. This is because the two-dimensional results are much more time consuming to compute than the one-dimensional results. The increase in α also happens much slower, which is due to the fact that the error threshold is reached much faster in the two-dimensional problem. This is explained by the fact that the error function evaluates the absolute, not the relative error, which is larger in two dimensions, since there are more points that contribute to the overall error.

6 Conclusion

In this report, first the equations describing multiscale particle dynamics have been discussed. Then the optimization framework has been introduced and first-order optimality systems have been derived for different test problems, which are simplified versions of the full problem for multiscale particle dynamics. The numerical methods, such as pseudospectral methods and multiple shooting have been introduced. Finally, some numerical experiments on the test problems provide evidence that the developed method produces good results for PDE-constrained optimization problems involving integro-PDE constraints.

The findings of this report are the first step towards building a numerical method that can be applied to the industrial applications and the particle systems that describe them. The next step in developing this method further is running more tests with different parameter values. For example, worth investigating is changing β, the regularization parameter, N, the number of spatial points, n, the number of time points, or ODE and `fsolve` tolerances. Further, the problem could be evaluated with different target distributions $\hat{\rho}$, as well as with attracting particles, rather than repulsive ones. Experimenting with different optimization solvers and ODE solvers is also a way to inform the further development of this method, as well as changing the continuation method for choosing α. Moreover, the efficiency of the code can be improved by parallelizing the calculations of the time intervals, and replacing some of the loops by matrix multiplications.

There are several ways of extending the numerical method, including applying different boundary conditions, adding external force terms, or different particle interactions, such as different choices of V_2 or the inclusion of hydrodynamic interactions. The latter, however, makes the problem significantly harder and therefore would be included at a later stage of the development process. The main aim for future work is to apply the numerical method to the industrial problems posed by WEST brewery and ufraction8. It is of interest to WEST how to optimize the sedimentation of yeast particles in beer, while ufraction8 would like to optimize their cell separation and nano-filtration devices.

References

[1] Jeremy Brandman, Huseyin Denli, and Dimitar Trenev. Introduction to PDE-constrained optimization in the oil and gas industry. In Harbir Antil, Drew P. Kouri, Martin-D. Lacasse, and Denis Ridzal, editors, *Frontiers in PDE-Constrained Optimization*, pages 171–203. Springer New York, New York, NY, 2018.

[2] W.F. Ramirez and J. Maciejowski. Optimal beer fermentation. *Journal of the Institute of Brewing*, 113(3):325–333, 2007.

[3] Juri Merger, Alfio Borzì, and Roland Herzog. Optimal control of a system of reaction–diffusion equations modeling the wine fermentation process. *Optimal Control Applications and Methods*, 38(1):112–132, 2017.

[4] Laura J. D. Frink, Andrew Salinger, Mark P. Sears, Jeffrey D. Weinhold, and Amalie L. Frischknecht. Numerical challenges in the application of density functional theory to biology and nanotechnology. *Journal of Physics: Condensed Matter*, 14:12167, 11 2002.

[5] Jianzhong Wu. Density functional theory for chemical engineering: From capillarity to soft materials. *AIChE Journal*, 52(3):1169–1193, 2006.

[6] M Rex and Hartmut Löwen. Dynamical density functional theory for colloidal dispersions including hydrodynamic interactions. *The European Physical Journal. E, Soft Matter*, 28:139–46, 2008.

[7] P. Tarazona, J.A. Cuesta, and Y. Martínez-Ratón. Density functional theories of hard particle systems. In Ángel Mulero, editor, *Theory and Simulation of Hard-Sphere Fluids and Related Systems*, pages 247–341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[8] Andreas Nold, Benjamin D. Goddard, Peter Yatsyshin, Nikos Savva, and Serafim Kalliadasis. Pseudospectral methods for density functional theory in bounded and unbounded domains. *CoRR*, abs/1701.06182, 2017.

[9] Arnold M. Arthurs. *Calculus of Variations*. Routledge and Keagan Paul Ltd, 1975.

[10] Fredi Tröltzsch. *Optimal Control of Partial Differential Equations: Theory, Methods and Applications*. Graduate Studies in Mathematics. American Mathematical Society, Providence, 2010.

[11] John W Pearson. *Fast Iterative Solvers for PDE-Constrained Optimization Problems*. PhD thesis, University of Oxford, 2013.

[12] Juan De los Reyes. *Numerical PDE-Constrained Optimization*. Springer, 2015.

[13] Enrique A. Gonzalez-Velasco. The product rule for Fréchet derivatives. *International Journal of Mathematical Education in Science and Technology*, 17:79–83, 1986.

[14] Thomas Carraro and Michael Geiger. Direct and indirect multiple shooting for parabolic optimal control problems. In P. Cinnella G. Zavarise and M. Campiti, editors, *Special Issue: 86th Annual Meeting of the International Association of Applied Mathematics and Mechanics (GAMM)*, pages 35–67. 01 2015.

[15] Lloyd N. Trefethen. *Spectral Methods in Matlab*. SIAM, 2000.

[16] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Review*, 46(3):501–517, 2004.

[17] John P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Inc, 2000.

[18] Charles W. Clenshaw and A. R. Curtis. A method for numerical integration on an automatic computer. *Numerische Mathematik*, 2(1):197–205, 1960.

[19] Wanfang Shen. Full-discrete adaptive fem for quasi-parabolic integro-differential pde-constrained optimal control problem. *Springer*, 2016.

[20] Stefan Güttel and John W. Pearson. A rational deferred correction approach to parabolic optimal control problems. *IMA Journal of Numerical Analysis*, 38:1861–1892, 2018.

[21] Jacek Kierzenka and Lawrence F. Shampine. A bvp solver based on residual control and the matlab pse. *ACM Transactions on Mathematical Software (TOMS)*, 27(3):299–316, 2001.

[22] Thomas Carraro, Michael Geiger, and R Rannacher. Indirect multiple shooting for nonlinear parabolic optimal control problems with control constraints. *SIAM Journal on Scientific Computing*, 36, 01 2014.

A One-Dimensional Problems Code

In this section the Matlab implementation in one dimension for the different stages of the numerical method can be found. Note that p is called q in the code.

A.1 Shooting Method

This script describes the setup of the space discretization, getting the initial condition, computing the ODE using `ode15s` and computing the error at the final time for p.

```
function data =ShootingTest_DiffusionLine1BlowsUp()
%Space Discretization
    N = 20;
    PhysArea = struct('shape','SpectralLine','N',N,'yMin',-1,'yMax',1);
    PlotArea = struct('N',200,'yMin',-1,'yMax',1);
    N = PhysArea.N;
    shapeClass = str2func(PhysArea.shape);
    aLine = shapeClass(PhysArea);
    [Pts,Diff,Int,Ind,~] = aLine.ComputeAll(PlotArea);
    DDyB = Diff.DDy;
    y = Pts.y;
    bound = Ind.bound;
    beta = 10^-3;
    tMax = 1;
    rhoMask = 1:N;
    qMask = N+1:2*N;
%Initial Conditions:
    rho_ic = rhoExact(y,0,tMax);
    q_ic = qExact(y,0,tMax);
    rhoq0=[rho_ic;q_ic];
% Output: Error in the final time for q
data=errorf(rhoq0);
    function err = errorf(rhoq0)
     odesol= odesolv(rhoq0);
     qsol=odesol(end,qMask);
     err=norm(qsol);
    end
   function odesol = odesolv(rhoq)
    mM   = ones(size(y));
    mM(bound)   = 0;
    mM = [mM;mM];
    opts = odeset('RelTol',10^-12,'AbsTol',10^-12,'Mass',diag(mM));

    [t,sol]= ode15s(@drhoq_dt, [0,0.1,1], rhoq,opts);
    odesol=sol;
   end
```

```
37      function drhoqdt = drhoq_dt(t,rhoq)
38       rho = rhoq(rhoMask);
39       q = rhoq(qMask);
40       rhoHat = rhoTarget(y,t,tMax);
41       drhodt  = DDyB*rho + (1/beta) * q;
42       dqdt =-DDyB*q+  rho - rhoHat;
43       %Dirichlet BCs
44       rhoExactBound = rhoExact(y(bound),t,tMax);
45       drhodt(bound) = rho(bound) - rhoExactBound;
46       qExactBound = qExact(y(bound),t,tMax);
47       dqdt(bound) = q(bound) - qExactBound;
48  %         %Neumann BCs
49  %           fluxrho = Diff.Dy*rho;
50  %           drhodt(bound) = fluxrho(bound);
51  %           fluxq = Diff.Dy*q;
52  %           dqdt(bound) = fluxq(bound);
53
54           drhoqdt = [drhodt;dqdt];
55      end
56       function rho = rhoExact(x,t,T)
57           rho = ( 4/(pi^2*beta)*exp(T) - 4/((4+pi^2)*beta)*exp(t) )*cos(pi*x/2);
58       end
59       function rho = rhoTarget(x,t,T)
60           rho = ( ( pi^2/4 + 4/(pi^2*beta) )*exp(T) ...
61                   + ( 1 - pi^2/4 - 4/((4+pi^2)*beta))*exp(t) ) *cos(pi*x/2);
62       end
63       function q = qExact(x,t,T)
64           q = (exp(T) - exp(t))*cos(pi*x/2);
65       end
66  end
```

A.2 Multiple Shooting, with the Exact Solution

This script discretizes time and computes the ODE solution on each time interval. It then calculates the error between the ODE solution and the initial value on each endpoint of the subintervals.

```
1  function data =DiffusionLine_MultipleShooting_ExactTest()
2   % Space Discretization as in A1.
3      Dirichlet=true;
4   % Time Discretization
5      TMaxG=1; % Choose global max
6      t0=0; %Choose global min
7      n=11; %Choose time steps
8      tMax = 1;
9      geom.yMin = t0;
```

```
10      geom.yMax = tMax ;
11      geom.N = n;
12      InterLine= SpectralLine(geom);
13      times = InterLine.Pts.y;
14      rhoMask = 1:N;
15      qMask = N+1:2*N;
16  for i=1:n-1
17      tMin=times(i);
18      tMax=times(i+1);
19      outTimes = [tMin: (tMax-tMin)/(n-1): tMax];
20      [odesolend,rhoq0]= Diffusion_1D_Solver(aLine,tMin,tMax,TMaxG,outTimes,
            Dirichlet);
21      finsol(i,:)=odesolend;
22      finIc(i,:)=rhoq0;
23  end
24  % %calculating error between current and previous time step in rho and q
25  % %(Simple approach):
26  % for j=1:9;
27  % rhoerror(j)= norm(finIc(j+1,rhoMask) - finsol(j,rhoMask));
28  % end
29  % for j=1:9;
30  % qerror(j)= norm(finIc(j,qMask) - finsol(j+1,qMask));
31  % end
32  % data=[rhoerror; qerror];
33  %calculating error between current and previous time step in rho and q for
34  %the tables
35  y=Pts.y;
36  beta=10^-3;
37      for i=1:length(times)
38          t=times(i);
39          if Dirichlet==true
40          rhoExa(i,:)=rhoExact(y,t,TMaxG);
41          qExa(i,:)=qExact(y,t,TMaxG);
42          else
43          rhoExa(i,:)=rhoExactN(y,t,TMaxG);
44          qExa(i,:)=qExactN(y,t,TMaxG);
45          end
46      end
47      if Dirichlet ==true
48      rho1=rhoExact(y,0,TMaxG)';    % True Initial Condition for rho
49      qEnd=qExact(y,TMaxG,TMaxG)'; % True Final Time Condition for q
50      else
51      rho1=rhoExactN(y,0,TMaxG)';
52      qEnd=qExactN(y,TMaxG,TMaxG)';
53      end
54      solEx=[rhoExa,qExa];
```

```
55      solEx1=solEx(2:n,rhoMask);
56      solEx2=solEx(1:n-1,qMask);
57      solExP=[solEx1,solEx2];
58      rhoError=(solExP(:,rhoMask) - finsol(:,rhoMask));
59      qError=(solExP(:,qMask) - finsol(:,qMask));
60      IntRhoErr=(((((Int*(rhoError.^2)').*(Int*(solExP(:,rhoMask).^2)'))))/(sqrt(N
            ))).^(1/2));
61      IntqErr=(((((Int*(qError.^2)').*(Int*(solExP(:,qMask).^2)'))))/(sqrt(N)))
            .^(1/2));
62      IntErr=[IntRhoErr,IntqErr];
63      data=max(IntErr);
64  % Exact solutions for Dirichlet BCs as in A1
65  % Exact solutions for Neumann BCs:
66      function rho = rhoExactN(x,t,T)
67          rho = ( 1/(pi^2*beta)*exp(T) - 1/((1+pi^2)*beta)*exp(t) )*cos(pi*x);
68      end
69      function rho = rhoTargetN(x,t,T)
70          rho = ( ( pi^2 + 1/(pi^2*beta) )*exp(T) ...
71              + ( 1 - pi^2 - 1/((1+pi^2)*beta))*exp(t) ) *cos(pi*x);
72      end
73      function q = qExactN(x,t,T)
74          q = (exp(T) - exp(t))*cos(pi*x);
75      end
76  end
```

This code solves the ODE using exact solutions for ρ and p.

```
1   function [odesolend,rhoq0]= Diffusion_1D_Solver(aLine,tMin,tMax,TMaxG,outTimes,
        Dirichlet)
2   % Space Discretization as in A1
3       %Initial Conditions:
4       if Dirichlet==true
5       rho_ic = rhoExact(y,tMin, TMaxG);
6       q_fin = qExact(y,tMax, TMaxG);
7       else
8       rho_ic = rhoExactN(y,tMin, TMaxG);
9       q_fin = qExactN(y,tMax, TMaxG);
10      end
11      rhoq0=[rho_ic;q_fin];
12      %computing the ODE solution and extracting rho and q:
13      odesol=odesolv(rhoq0);
14      odesolend=odesol(end,:);
15      rhosol=odesol(end,rhoMask);
16      qsol=odesol(end,qMask);
17     function odesol = odesolv(rhoq)
18          mM   = ones(size(y));
19          mM(bound)   = 0;
```

```
20          mM = [mM;mM];
21          opts = odeset('RelTol',10^-12,'AbsTol',10^-12,'Mass',diag(mM));
22          [t,sol]= ode15s(@drhoq_dt, outTimes, rhoq, opts);
23          odesol=sol;
24     end
25 function drhoqdt = drhoq_dt(t,rhoq)
26          rho = rhoq(rhoMask);
27          q = rhoq(qMask);
28 if Dirichlet==true
29          rhoHat = rhoTarget(y,  tMax-t+tMin, TMaxG);
30          rhoEx= rhoExact(y,  tMax-t+tMin, TMaxG);
31          qEx=qExact(y,t, TMaxG);
32 else
33          rhoHat = rhoTargetN(y,  tMax-t+tMin, TMaxG);
34          rhoEx= rhoExactN(y,  tMax-t+tMin, TMaxG);
35          qEx=qExactN(y,t, TMaxG);
36 end
37          drhodt  = DDyB*rho + (1/beta) * qEx;
38          dqdt =DDyB*q+ rhoHat -rhoEx ;
39   if Dirichlet==true
40        rhoExactBound = rhoExact(y(bound),t, TMaxG);
41        drhodt(bound) = rho(bound) - rhoExactBound;
42        qExactBound = qExact(y(bound),tMax-t+tMin, TMaxG);
43        dqdt(bound) = q(bound) - qExactBound;
44   else
45          fluxrho =DyB*rho;
46          drhodt(bound) = fluxrho(bound);
47          fluxq = DyB*q;
48          dqdt(bound) = fluxq(bound);
49   end
50          drhoqdt =[drhodt;dqdt];
51    end
52 % Exact Solutions as in A1.
53 end
```

A.3 Multiple Shooting, with Interpolation on an Equispaced Grid

This code computes the ODE using interpolation onto a finer, equispaced grid. The error beween the ODE solution when using interpolation and when using exact values is measured in the L^2 norm in space and the L^∞ norm in time.

```
1 function data =DiffusionLine_Interpol_Equispaced()
2 % Spatial and Time Discretization as in A1
3 % Getting rho and q on chebyshev points. (Initial 'Guess') as in A2
4     % Interpolate time for each space point y
5          solInt = InterpM * solEx;
```

```
6       % Extract rho, q interpolated
7           rhoInt=solInt(:,rhoMask);
8           qInt=solInt(:,qMask);
9       % Calculating Interpolation Error
10          rhoErr= norm(rhoInt-rhoExI);
11          qErr= norm(qInt-qExI);
12          InterpErr=[rhoErr,qErr];
13  %Initial conditions for rho and q as in A2
14  % Calculating Interpolation Error
15      rhoErr=(rhoInt-rhoExI);
16      qErr=(qInt-qExI);
17      ODEerr=(odesolvEX(rhoq0)-odesolv(rhoq0));
18      ODEEx=odesolvEX(rhoq0);
19      IntRhoErr=(((((Int*(rhoErr.^2)').*(Int*(rhoExI.^2)')))/(sqrt(N))).^(1/2));
20      IntqErr=(((((Int*(qErr.^2)').*(Int*(qExI.^2)')))/(sqrt(N))).^(1/2));
21      IntODEErrRho=(((((Int*(ODEerr(:,rhoMask).^2)').*(Int*(ODEEx(:,rhoMask).^2)
            ')))/(sqrt(N))).^(1/2));
22      IntODEErrq=(((((Int*(ODEerr(:,qMask).^2)').*(Int*(ODEEx(:,qMask).^2)')))/(
            sqrt(N))).^(1/2));
23      IntODEErr=[ IntODEErrRho, IntODEErrq];
24      data.a=max(IntRhoErr);
25      data.b=max(IntqErr);
26      data.c=max(IntODEErr);
27      function odesol = odesolv(rhoq)
28       % As in A2
29      end
30     function drhoqdt = drhoq_dt(t,rhoq)
31          [~,tPos]=min(abs(t-tvec));
32          tau=tvec(tPos);
33          [~,tPosFlip]=min(abs((tMax-t)-tvec));
34          tflip = tvec(tPosFlip);
35          rho = rhoq(rhoMask);
36          q = rhoq(qMask);
37  if Dirichlet == true
38          rhoHat = rhoTarget(y,tflip,tMax);
39  else
40          rhoHat = rhoTargetN(y,tflip,tMax);
41  end
42          rhoI=rhoInt(tPosFlip,:)';
43          qI=qInt(tPos,:)';
44          drhodt  = DDyB*rho + (1/beta) * qI;
45          dqdt =DDyB*q-  rhoI + rhoHat;
46  if Dirichlet==true
47          rhoExactBound = rhoExact(y(bound),tau,tMax);
48          drhodt(bound) = rho(bound) - rhoExactBound;
49          qExactBound = qExact(y(bound),tflip,tMax);
```

```
50          dqdt(bound) = q(bound) - qExactBound;
51  else
52           fluxrho = Diff.Dy*rho;
53           drhodt(bound) = fluxrho(bound);
54           fluxq = Diff.Dy*q;
55           dqdt(bound) = fluxq(bound);
56  end
57
58          drhoqdt = [drhodt;dqdt];
59     end
60  %%% exact ode solver
61      function odesol = odesolvEX(rhoq)
62      % As in A2
63      end
64      function drhoqdt = drhoq_dtEX(t,rhoq)
65      % As in A2
66      end
67  %Exact solutions as in A1 and A2.
68  end
```

A.4 Multiple Shooting, with Chebyshev Interpolation

This code computes the ODE using pointwise Chebyshev interpolation. The error to the ODE solution when using exact values is measured in the L^2 norm in space and L^∞ norm in time.

```
1   function data =DiffusionLine_Interpol_Chebyshev()
2   % Spatial and Time Discretization as in A1
3   % Getting rho and q on chebyshev points. (Initial 'Guess') as in A2
4   % Calculating Interpolation Error
5       ODEerr=(odesolvEX(rhoq0)-odesolv(rhoq0));
6       ODEEx=odesolvEX(rhoq0);
7       IntODEErrRho=(((((Int*(ODEerr(:,rhoMask).^2)').*(Int*(ODEEx(:,rhoMask).^2)
            '))))/(sqrt(N))).^(1/2));
8       IntODEErrq=(((((Int*(ODEerr(:,qMask).^2)').*(Int*(ODEEx(:,qMask).^2)'))))/(
            sqrt(N))).^(1/2));
9       IntODEErr=[ IntODEErrRho, IntODEErrq];
10      data=max(IntODEErr);
11      function odesol = odesolv(rhoq)
12      % As in A2
13      end
14     function drhoqdt = drhoq_dt(t,rhoq)
15          rho = rhoq(rhoMask);
16          q = rhoq(qMask);
17          InterpT = InterLine.ComputeInterpolationMatrix(InterLine.CompSpace([t;
                tMax+t0-t]));
18          Interp=InterpT.InterPol;
```

```
19          rhoIn=Interp(2,:)*rhoExa;
20          rhoI=rhoIn';
21          qIn=Interp(1,:)*qExa;
22          qI=qIn';
23         if Dirichlet == true
24           rhoHat = rhoTarget(y,tMax+t0-t,tMax);
25         else
26           rhoHat = rhoTargetN(y,tMax+t0-t,tMax);
27         end
28          drhodt  = DDyB*rho + (1/beta) * qI;
29          dqdt =DDyB*q -  rhoI + rhoHat;
30 if Dirichlet == true
31          rhoExactBound = rhoExact(y(bound),t,tMax);
32          drhodt(bound) = rho(bound) - rhoExactBound;
33          qExactBound = qExact(y(bound),tMax+t0-t,tMax);
34          dqdt(bound) = q(bound) - qExactBound;
35 else
36           fluxrho = Diff.Dy*rho;
37           drhodt(bound) = fluxrho(bound);
38           fluxq = Diff.Dy*q;
39           dqdt(bound) = fluxq(bound);
40 end
41          drhoqdt = [drhodt;dqdt];
42    end
43 %%% exact ode solver
44     function odesol = odesolvEX(rhoq)
45   % As in A2
46     end
47    function drhoqdt = drhoq_dtEX(t,rhoq)
48   % As 'drhoq_dt(t,rhoq)' in A2
49    end
50 % Exact solutions as in A1 and A2.
51 end
```

A.5 Multiple Shooting, with Interpolation and Optimization of Initial Guess

This script is computing the error on the t_i, minimizing the error function with `fsolve`.

```
1 function data =DiffusionLine_MultipleShooting_WInterp_FsolveSplit(params,~)
2 % Setting up Spatial and time discretization as in A1 and A2, getting Initial
     guess as in A2.
3 % Solving the minimization problem (giving it perturbed Initial Guess)
4     options = optimoptions('fsolve','FunctionTolerance',FunTol,'
          OptimalityTolerance',OptiTol,'StepTolerance',StepTol,'Display','iter');
5     [OptiIc,~,~,output]=fsolve(@errorf,solExP,options);
6     solEx5=[rho1;OptiIc(:,rhoMask)];
```

```
7       solEx6=[OptiIc(:,qMask);qEnd];
8       OptiIcG=[solEx5,solEx6];
9   % Output
10  data.solution=OptiIcG;
11  data.error= norm(OptiIcG-solEx);
12  data.aLine=aLine;
13  data.InterLine=InterLine;
14  data.output=output;
15  % Function calculating error between current and previous time step in rho and
        q
16      function err =errorf(solExP)
17              solEx3=[rho1;solExP(:,rhoMask)];
18              solEx4=[solExP(:,qMask);qEnd];
19              solExN=[solEx3,solEx4];
20          %Solving Multiple Shooting for a given Initial Guess
21              [FinSol]= MultipleShooting_Solver(aLine,InterLine,TMaxG,solExN,N,
                    beta,Dirichlet,ODEtols);
22          %Calculating the Error
23              rhoError=(solExP(:,rhoMask) - FinSol(:,rhoMask));
24              qError=(solExP(:,qMask) - FinSol(:,qMask));
25              err=[rhoError;qError];
26      end
27  end
```

This script computes the different intervals, passes these to the ODE solver and outputs ODE solutions on all intervals.

```
1   function [FinSol]= MultipleShooting_Solver(aLine,InterLine,TMaxG,solExN,N,beta,
        Dirichlet,ODEtols)
2       % Compare to A2.
3       rhoMask = 1:N;
4       qMask = N+1:2*N;
5       times = InterLine.Pts.y;
6       n=length(times);
7   for i=1:n-1
8           tMin=times(i);
9           tMax=times(i+1);
10          outTimes = [tMin: (tMax-tMin)/(n-1): tMax];
11      %Initial conditions for rho and q (from interpolation matrix)
12          rho_ic=solExN(i,rhoMask);
13          q_ic=solExN(i+1,qMask);
14          rhoq0= [rho_ic, q_ic];
15          FinIcP(i,:)=rhoq0;
16          [odesolend]= Diffusion_1D_Solver_WInterp(aLine,InterLine,tMin,tMax,
                TMaxG,outTimes,solExN,rhoq0,beta,Dirichlet,ODEtols);
17          FinSol(i,:)=odesolend;
18  end
```

```
19  end
```

This script solves the ODE using interpolation for ρ and p.

```
function [odesolend]= Diffusion_1D_Solver_WInterp(aLine,InterLine,tMin,tMax,
    TMaxG,outTimes,solExN,rhoq0,beta,Dirichlet,ODEtols);
     function odesol = odesolv(rhoq)
    % As in A2
     end
    function drhoqdt = drhoq_dt(t,rhoq)
    % As in A4
    end
end
```

A.6 Perturbing the Initial Guess and Data Storage

This code is setting up the data storage function and the perturbation routine using perturbation functions. Then the optimization and ODE solution is calculated using the code in Appendix A.5.

```
function Data=DataStorageMultipleShootingFunctPert()
% Defining Parameters
    params.N = 20;
    params.yMin = -1;
    params.yMax = 1;
    params.t0=0;
    params.TMaxG = 1;
    params.n = 11;
    params.beta = 10^(-3);
    params.Dirichlet=false;
    params.Deterministic=false;
    params.fsolvetols.FunTol=10^(-6);
    params.fsolvetols.OptiTol=0.1;
    params.fsolvetols.StepTol=10^(-6);
    params.ODEtols.RelTol=10^-8;
    params.ODEtols.AbsTol=10^-8;
% Creating Space and time discretization as in A1 and A2
% Initial guess creation as in A2
% Pertubation vector to perturb initial guess:
     rhoper=[0,0.01,0.1,1,0,0.01,0.1,1,0,0.01,0.1,1,0,0.01,0.1,1,0.01,0.1,1];
     qper=[0,0,0,0,0.01,0.01,0.01,0.01,0.1,0.1,0.1,0.1,1,1,1,1,0.01,0.1,1];
     tpmask=[0,ones(1,n-3),0; zeros(1,n-6),ones(1,4),0; zeros(1,n-3),1,0;zeros
         (1,n-6),1,zeros(1,4);
         0,ones(1,n-3),0; zeros(1,n-6),ones(1,4),0; zeros(1,n-3),1,0; zeros(1,n
             -6),1,zeros(1,4);
         0,ones(1,n-3),0;zeros(1,n-6),ones(1,4),0;zeros(1,n-3),1,0;zeros(1,n-6)
             ,1,zeros(1,4);
```

```
25          0,ones(1,n-3),0;zeros(1,n-6),ones(1,4),0;zeros(1,n-3),1,0;zeros(1,n-6)
                ,1,zeros(1,4);
26          0,ones(1,n-3),0;0,ones(1,n-3),0;0,ones(1,n-3),0];
27      rhopmask=[ones(1,N-2); ones(1,N-2); ones(1,N-2);ones(1,N-2);
28          zeros(1,N-7),ones(1,5);zeros(1,N-7),ones(1,5);zeros(1,N-7),ones(1,5);
                zeros(1,N-7),ones(1,5);
29          zeros(1,N-3),1;zeros(1,N-3),1;zeros(1,N-3),1;zeros(1,N-3),1;zeros(1,N
                -7),1,zeros(1,4);
30          zeros(1,N-7),1,zeros(1,4);zeros(1,N-7),1,zeros(1,4);zeros(1,N-7),1,
                zeros(1,4);
31          ones(1,N-2);ones(1,N-2);ones(1,N-2)];
32      qpmask=[ones(1,N-2);ones(1,N-2);ones(1,N-2);ones(1,N-2);
33          zeros(1,N-7),ones(1,5);zeros(1,N-7),ones(1,5);zeros(1,N-7),ones(1,5);
                zeros(1,N-7),ones(1,5);
34          zeros(1,N-3),1;zeros(1,N-3),1;zeros(1,N-3),1;zeros(1,N-3),1;
35          zeros(1,N-7),1,zeros(1,4);zeros(1,N-7),1,zeros(1,4);zeros(1,N-7),1,
                zeros(1,4);zeros(1,N-7),1,zeros(1,4);
36          ones(1,N-2);ones(1,N-2);ones(1,N-2)];
37  for i=1:19
38      pparams.rhoper=rhoper(i);pparams.qper=qper(i);pparams.tpmask=tpmask(i,:);
39      pparams.rhopmask=rhopmask(i,:);pparams.qpmask=qpmask(i,:);
40      pparams.solExP=solExP;params.pparams=pparams;
41      if Deterministic == true
42      params.IGPert=pertfun(pparams);
43      else
44      params.IGPert=pertfunR(pparams);
45      end
46      dataDir = 'MultipleShooting';
47      recompute = false;
48  % Compute solution use function in A5
49   Data(i) = DataStorage(dataDir,
        @DiffusionLine_MultipleShooting_WInterp_FsolveSplitFunctPert,params,[],
        recompute);
50  end
51  %Defining the location of the pertubation (Deterministic)
52      function IGPert = pertfun(pparams)
53          rhoper1=pparams.rhoper;
54          qper1=pparams.qper;
55          tpert1=pparams.tpmask;
56          rhopmask1=pparams.rhopmask;
57          qpmask1=pparams.qpmask;
58          solExP=pparams.solExP;
59          % Perturb the required points in rho and q by reuired amount
60          % Add zeros on both ends, then stack
61          rhomp=[0,rhopmask1*rhoper1,0];
62          qmp=[0,qpmask1*qper1,0];
```

```
63          pert1=[rhomp,qmp];
64          % Create pertubation mask and apply to the exact solution
65          Pert=(ones(size(solExP)).*tpert1'.*pert1);
66          IGPert=solExP + Pert;
67      end
68  %Defining the location of the pertubation (Random)
69      function IGPert = pertfunR(pparams)
70          rhoper1=pparams.rhoper;
71          qper1=pparams.qper;
72          tpert1=pparams.tpmask;
73          rhopmask1=pparams.rhopmask;
74          qpmask1=pparams.qpmask;
75          solExP=pparams.solExP;
76          % Create two random vectors and multiply by pertubation strength
77          rng(1);
78          rr=rand(1,N-2)*rhoper1;
79          rng(2);
80          rq=rand(1,N-2)*qper1;
81          % Perturb and stack (add zeros to padd)
82          rhomp=[0,rhopmask1.*rr,0];
83          qmp=[0,qpmask1.*rq,0];
84          pert1=[rhomp,qmp];
85          % Create perturbation mask and apply to exact solution
86          Pert=(ones(size(solExP)).*tpert1'.*pert1);
87          IGPert=solExP + Pert;
88      end
89  end
```

A.7 Continuation in One Dimension

This code computes the continuation method described in the numerics section. Note that α in the text is the variable *gam* in the code.

```
1  function Data=DataStorageMultipleShootingFunctPertConv()
2  AddPaths;
3  % Setup as in A6
4      beta=params.beta;
5      alpha=1;
6      ConvV=ComputeConvolutionMatrix(SpaceBox,@Gauss,true);
7      params.ConvV=ConvV;
8  % Continuation:
9   i=0;
10  gam=0.0001;
11  NewIG=solExP;
12 while abs(gam) <0.1
13 i=i+1
```

```
14 IGError=MultipleShootingError(aLine,InterLine,TMaxG,NewIG,rho1,qEnd,N,beta,
       Dirichlet,ODEtols,ConvV,gam);
15 if abs(gam)<0.05
16         while norm(IGError) < 10^3
17         gam=gam*1.1
18         IGError=MultipleShootingError(aLine,InterLine,TMaxG,NewIG,rho1,qEnd,N,
               beta,Dirichlet,ODEtols,ConvV,gam);
19         end
20   elseif abs(gam)<0.65
21         while norm(IGError) < 10^2
22         gam=gam*1.01
23         IGError=MultipleShootingError(aLine,InterLine,TMaxG,NewIG,rho1,qEnd,N,
               beta,Dirichlet,ODEtols,ConvV,gam);
24         end
25   else
26         while norm(IGError) < 50
27         gam=gam*1.001
28         IGError=MultipleShootingError(aLine,InterLine,TMaxG,NewIG,rho1,qEnd,N,
               beta,Dirichlet,ODEtols,ConvV,gam);
29         end
30 end
31  params.IGPert=NewIG;
32  params.gam=gam;
33  Data = DataStorage(dataDir,
       @DiffusionLine_MultipleShooting_WInterp_FsolveSplitFunctPertConv,params,[],
       recompute);
34  NewIG=Data.solution;
35  Iters=Data.output.iterations
36  if Iters<20
37      if abs(gam)<0.05
38           gam=gam*1.1
39      elseif abs(gam)<0.65
40          gam=gam*1.01
41      else
42          gam=gam*1.001
43      end
44  else
45      return
46  end
47 end
48     function g= Gauss(y)
49         g= alpha* exp(-y.^2);
50     end
51 end
```

This script computes the error function.

```
function  error= MultipleShootingError(aLine,InterLine,TMaxG,IGPert,rho1,qEnd,N
    ,beta,Dirichlet,ODEtols,ConvV,gam);
        rhoMask = 1:N;
        qMask = N+1:2*N;
        solEx3=[rho1;IGPert(:,rhoMask)];
        solEx4=[IGPert(:,qMask);qEnd];
        solExN=[solEx3,solEx4];
        [FinSol]= MultipleShooting_SolverConv(aLine,InterLine,TMaxG,solExN,N,
            beta,Dirichlet,ODEtols,ConvV,gam);
        rhoError=(IGPert(:,rhoMask) - FinSol(:,rhoMask));
        qError=(IGPert(:,qMask) - FinSol(:,qMask));
        error=[rhoError;qError];
end
```

This script computes the ODE solution including the integral term.

```
function [odesolend]= Diffusion_1D_Solver_WInterpConv(aLine,InterLine,tMin,tMax
    ,TMaxG,outTimes,solExN,rhoq0,beta,Dirichlet,ODEtols,ConvV,gam);
    function odesol = odesolv(rhoq)
     % As in A2
    end
   function drhoqdt = drhoq_dt(t,rhoq)
        rho = rhoq(rhoMask);
        q = rhoq(qMask);
        InterpT = InterLine.ComputeInterpolationMatrix(InterLine.CompSpace([t;
            tMax+tMin-t]));
        Interp=InterpT.InterPol;
        rhoIn=Interp(2,:)*rhoExa;
        rhoI=rhoIn';
        qIn=Interp(1,:)*qExa;
        qI=qIn';
        Dyq=Dy*q;
        rhoHat = rhoTarget(y,tMax+tMin-t,TMaxG);
        fluxrho =-(Dy*rho + gam*rho.*(Dy*(ConvV*rho)));
        drhodt  = - Dy*fluxrho + (1/beta) * qI;
        dqdt =DDy*q -  rhoI + rhoHat - gam*(((Dyq).* (Dy*(ConvV*rhoI)))+ (Dy*(
            ConvV*(rhoI.*(Dyq)))));
        rhoExactBound = rhoExact(y(bound),t,TMaxG);
        drhodt(bound) = rho(bound) - rhoExactBound;
        qExactBound = qExact(y(bound),tMax+tMin-t,TMaxG);
        dqdt(bound) = q(bound) - qExactBound;
        drhoqdt = [drhodt;dqdt];
   end
end
```

B Two-Dimensional Problems Code

In this section the Matlab implementation in two dimensions for the different stages of the numerical method can be found.

B.1 Multiple Shooting in Two Dimensions

Equivalently to the one-dimensional code, this script computes data storage and perturbation functions for two-dimensional problems.

```
function Data=DataStorageMultipleShootingFunctPert2D()
% Defining Parameters as in A6
% Creating the lines
% Spatial Chebyshev points 2D
    N1=params.N1;
    N2=params.N2;
    geom.y1Min = params.geom.y1Min;
    geom.y1Max = params.geom.y1Max;
    geom.y2Min = params.geom.y2Min;
    geom.y2Max = params.geom.y2Max;
    geom.N = [N1;N2];
    % make the box
    SpaceBox = Box(geom);
    % compute points and matrices
    [Pts,Diff,Int,Ind] = SpaceBox.ComputeAll();
    beta=params.beta;
    % Kronecker space vectors, length N1xN2
    y1_kv=SpaceBox.Pts.y1_kv;
    y2_kv=SpaceBox.Pts.y2_kv;
    rhoMask = 1:length(y1_kv);
    qMask = (length(y1_kv)+1):(2*length(y1_kv));
% Creating a line in t as in 1D
%Creating an Initial Guess as in 1D but taking the exact solution in 2D
% Compute solution
%Pertubation vector
N=100;
     rhoper=[0,0.01,0.1,1,0.01,0.01,0.1,1,0.1];
     qper=[0,0,0,0,0,0.01,0.1,1,0.01];
     tpmask=[0,ones(1,n-3),0;zeros(1,n-6),ones(1,4),0;zeros(1,n-3),1,0; zeros
         (1,n-6),1,zeros(1,4);
         0,ones(1,n-3),0;0,ones(1,n-3),0; 0,ones(1,n-3),0;0,ones(1,n-3),0;0,
             ones(1,n-3),0];
     rhopmask=[ones(1,N-2);zeros(1,12),1,zeros(1,N-15); zeros(1,22),1,zeros(1,N
         -25);zeros(1,32),1,zeros(1,N-35);
         zeros(1,9),0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,
             ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,zeros(1,9);
```

```
33          zeros(1,12),1,zeros(1,N-15);zeros(1,22),1,zeros(1,N-25); zeros(1,32)
                ,1,zeros(1,N-35);
34          zeros(1,9),0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,
                ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,zeros(1,9)];
35      qpmask=[ones(1,N-2);ones(1,N-2);ones(1,N-2);ones(1,N-2);ones(1,N-2);
36          zeros(1,12),1,zeros(1,N-15);zeros(1,22),1,zeros(1,N-25); zeros(1,32)
                ,1,zeros(1,N-35);
37          zeros(1,9),0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,
                ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,0,ones(1,8),0,zeros(1,9)];
38  for i=1:9
39      pparams.rhoper=rhoper(i);pparams.qper=qper(i);
40      pparams.tpmask=tpmask(i,:);pparams.rhopmask=rhopmask(i,:);
41      pparams.qpmask=qpmask(i,:);pparams.solExP=solExP; params.pparams=pparams;
42  if Deterministic == true
43       params.IGPert=pertfun(pparams);
44  else
45      params.IGPert=pertfunR(pparams);
46  end
47      dataDir = 'MultipleShooting';
48      recompute = false;
49  % Compute solution, called function see 1D case.
50      Data(i) = DataStorage(dataDir,
            @DiffusionLine_MultipleShooting_WInterp_FsolveSplitFunctPert2D,params
            ,[],recompute);
51  end
52  %Defining the location of the pertubation
53      function IGPert = pertfun(pparams)
54        % As in A6
55      end
56  %Defining the location of the pertubation (+random)
57      function IGPert = pertfunR(pparams)
58        % As in A6
59      end
60  end
```

This code contains the ODE solver for two-dimensional problem.

```
1   function [odesolend]= Diffusion_2D_Solver_WInterp(SpaceBox,InterLine,tMin,tMax,
        TMaxG,outTimes,solExN,rhoq0,beta,Dirichlet,ODEtols);
2       RelTol=ODEtols.RelTol;
3       AbsTol=ODEtols.AbsTol;
4       N1 = SpaceBox.N1;
5       N2=SpaceBox.N2;
6       Lap= SpaceBox.Diff.Lap;
7       Grad = SpaceBox.Diff.grad;
8       bound = SpaceBox.Ind.bound;
9       y1_kv=SpaceBox.Pts.y1_kv;
```

```
10      y2_kv=SpaceBox.Pts.y2_kv;
11      rhoMask = 1:length(y1_kv);
12      qMask = (length(y1_kv)+1):(2*length(y1_kv));
13  %Extracting exact rho,q on Chebyshev points from solEx
14      rhoExa=solExN(:,rhoMask);
15      qExa=solExN(:,qMask);
16  %computing the ODE solution and extracting rho and q:
17      odesol=odesolv(rhoq0);
18      odesolend=odesol(end,:);
19      rhosol=odesol(end,rhoMask);
20      qsol=odesol(end,qMask);
21      function odesol = odesolv(rhoq)
22          % As in A6
23      end
24     function drhoqdt = drhoq_dt(t,rhoq)
25          rho = rhoq(rhoMask);
26          q = rhoq(qMask);
27          InterpT = InterLine.ComputeInterpolationMatrix(InterLine.CompSpace([t;
                tMax+tMin-t]));
28          Interp=InterpT.InterPol;
29          rhoIn=Interp(2,:)*rhoExa;
30          rhoI=rhoIn';
31          qIn=Interp(1,:)*qExa;
32          qI=qIn';
33       if Dirichlet == true;
34          rhoHat = rhoTarget(y1_kv,y2_kv,tMax+tMin-t,TMaxG);
35       else
36          rhoHat = rhoTargetN(y1_kv,y2_kv,tMax+tMin-t,TMaxG);
37       end
38     % Two Dimensional Optimality System
39          drhodt  = Lap*rho + (1/beta) * qI;
40          dqdt =Lap*q -  rhoI + rhoHat;
41     % Two Dimensional BCs
42        if Dirichlet==true
43                  rhoExactBound = rhoExact(y1_kv(bound),y2_kv(bound),t,TMaxG);
44                  drhodt(bound) = rho(bound) - rhoExactBound;
45                  qExactBound = qExact(y1_kv(bound),y2_kv(bound),tMax+tMin-t,
                        TMaxG);
46                  dqdt(bound) = q(bound) - qExactBound;
47        else
48                   fluxrho =Grad*rho;
49                   drhodt(bound) = fluxrho(bound);
50                   fluxq = Grad*q;
51                   dqdt(bound) = fluxq(bound);
52        end
53          drhoqdt = [drhodt;dqdt];
```

```
54     end
55 % Two Dimensional Exact Solutions
56   function rho = rhoExact(x1,x2,t,T)
57         rho = ( 4/(2*pi^2*beta)*exp(T) - 4/((4+2*pi^2)*beta)*exp(t) )*cos(pi*x1
               /2).*cos(pi*x2/2);
58     end
59     function rho = rhoTarget(x1,x2,t,T)
60         rho = ( ( 2*pi^2/4 + 4/(2*pi^2*beta) )*exp(T) ...
61                 + ( 1 - 2*pi^2/4 - 4/((4+2*pi^2)*beta))*exp(t) ) *cos(pi*x1/2)
                       .*cos(pi*x2/2);
62     end
63  function q = qExact(x1,x2,t,T)
64         q = (exp(T) - exp(t))*cos(pi*x1/2).*cos(pi*x2/2);
65  end
66   function rho = rhoExactN(x1,x2,t,T)
67         rho = ( 1/(2*pi^2*beta)*exp(T) - 1/((1+2*pi^2)*beta)*exp(t) )*cos(pi*x1
               ).*cos(pi*x2);
68     end
69     function rho = rhoTargetN(x1,x2,t,T)
70         rho = ( ( 2*pi^2 + 1/(pi^2*beta) )*exp(T) ...
71                 + ( 1 - 2*pi^2 - 1/((1+2*pi^2)*beta))*exp(t) ) *cos(pi*x1).*cos
                       (pi*x2);
72     end
73  function q = qExactN(x1,x2,t,T)
74         q = (exp(T) - exp(t))*cos(pi*x1).*cos(pi*x2);
75  end
76 end
```

B.2 Continuation in Two Dimensions

Similar to the one-dimensional code, this script contains the continuation method in 2D. This script computes the continuation method using the code for the `fsolve` routine as well as the error function, as described in Appendix A.7.

```
1 function Data=DataStorageMultipleShooting2DConv()
2 % Setup as in B1
3     i=0;
4     gam=-0.0001;
5     NewIG=solExP;
6 while abs(gam) <0.1
7 i=i+1
8 IGError=MultipleShootingError2D(SpaceBox,InterLine,TMaxG,NewIG,rho1,qEnd,beta,
      Dirichlet,ODEtols,ConvV,gam);
9 if abs(gam)<0.05
10         while norm(IGError) < 10^2
11         gam=gam*1.1
```

```
12          IGError=MultipleShootingError2D(SpaceBox,InterLine,TMaxG,NewIG,rho1,
                qEnd,beta,Dirichlet,ODEtols,ConvV,gam);
13          end
14    elseif abs(gam)<0.65
15          while norm(IGError) < 10^2
16          gam=gam*1.01
17          IGError=MultipleShootingError2D(SpaceBox,InterLine,TMaxG,NewIG,rho1,
                qEnd,beta,Dirichlet,ODEtols,ConvV,gam);
18          end
19    else
20          while norm(IGError) < 50
21          gam=gam*1.001
22          IGError=MultipleShootingError2D(SpaceBox,InterLine,TMaxG,NewIG,rho1,
                qEnd,beta,Dirichlet,ODEtols,ConvV,gam);
23          end
24  end
25   params.IGPert=NewIG;
26   params.gam=gam;
27   Data = DataStorage(dataDir,
        @DiffusionLine_MultipleShootingFsolveSplitFunctPert2DConv,params,[],
        recompute);
28   NewIG=Data.solution;
29   Iters=Data.output.iterations
30   if Iters<20
31       if abs(gam)<0.05
32            gam=gam*1.1
33       elseif abs(gam)<0.65
34           gam=gam*1.01
35       else
36           gam=gam*1.001
37       end
38   else
39       return
40   end
41  end
42  % Two Dimensional Gaussian
43   function g= Gauss(y1,y2)
44          g= alpha* exp(-(y1.^2+y2.^2));
45   end
46  end
```

This code computes the ODE in two dimension including the non-local term.

```
1  function [odesolend]= Diffusion_2D_Solver_WInterpConv(SpaceBox,InterLine,tMin,
       tMax,TMaxG,outTimes,solExN,rhoq0,beta,Dirichlet,ODEtols,ConvV,gam);
2  % Setup as in B1
3      function odesol = odesolv(rhoq)
```

```
4         % As in A2
5      end
6     function drhoqdt = drhoq_dt(t,rhoq)
7          rho = rhoq(rhoMask);
8          q = rhoq(qMask);
9          InterpT = InterLine.ComputeInterpolationMatrix(InterLine.CompSpace([t;
               tMax+tMin-t]));
10         Interp=InterpT.InterPol;
11         rhoIn=Interp(2,:)*rhoExa;
12         rhoI=rhoIn';
13         qIn=Interp(1,:)*qExa;
14         qI=qIn';
15         rhoHat = rhoTarget(y1_kv,y2_kv,tMax+tMin-t,TMaxG);
16         % Computing the integral terms
17         Conv1 = ConvV*((Grad1*q).*rhoI);
18         Conv2 = ConvV*((Grad2*q).*rhoI);
19         Int1= Grad1*Conv1 + Grad2*Conv2;
20         Int2= dotProduct(Grad*q, (Grad*ConvV)*rhoI);
21         fluxrho =-(Grad*rho + gam*Grad*(rho.*(ConvV*rho)));
22         drhodt  = - Div*fluxrho + (1/beta) * qI;
23         dqdt =Lap*q -  rhoI + rhoHat - gam*(Int1 +Int2);
24         rhoExactBound = rhoExact(y1_kv(bound),y2_kv(bound),t,TMaxG);
25         drhodt(bound) = rho(bound) - rhoExactBound;
26         qExactBound = qExact(y1_kv(bound),y2_kv(bound),tMax+tMin-t,TMaxG);
27         dqdt(bound) = q(bound) - qExactBound;
28         drhoqdt = [drhodt;dqdt];
29    end
30 % Dot product function
31     function dprod= dotProduct(u,v)
32         dprod= u(1:end/2).*v(1:end/2) + u(end/2+1:end).*v(end/2+1:end);
33     end
34 end
```

C Tables

Some larger tables can be found in this Appendix.

C.1 Perturbing the Exact Solution in One Dimension

The two tables in this Appendix show the perturbation locations and strengths, the error with respect to the exact solution and the number of function evaluation `fsolve` did before finding the optimal initial guess. The results for Dirichlet boundary conditions can be found in Table 8 and for Neumann boundary conditions in Table 9.

Perturbation Location			Perturbation		Deterministic Pert.		Random Pert.	
ρ	q	t	ρ	q	Error	F.Eval.	Error	F.Eval.
none	none	none	0	0	0	401	0	401
all	none	4	0.01	0	4.8417×10^{-7}	1219	5.7913×10^{-7}	817
all	none	1 (e)	0.1	0	4.1715×10^{-7}	802	4.6127×10^{-8}	802
all	none	1 (m)	1	0	5.3019×10^{-6}	1621	2.2027×10^{-7}	1203
none	4	all	0	0.01	5.2417×10^{-7}	802	1.5254×10^{-7}	802
4	4	4	0.01	0.01	1.1291×10^{-6}	802	4.6255×10^{-7}	802
4	4	1 (e)	0.1	0.01	3.2506×10^{-7}	802	1.6869×10^{-7}	802
4	4	1 (m)	1	0.01	1.3258×10^{-7}	1203	1.0726×10^{-7}	1203
none	1	all	0	0.1	2.5343×10^{-6}	802	4.3011×10^{-7}	802
1(e)	1(e)	4	0.01	0.1	4.5179×10^{-7}	802	1.5792×10^{-7}	802
1 (e)	1 (e)	1 (e)	0.1	0.1	1.5305×10^{-7}	802	6.5447×10^{-8}	802
1 (e)	1(e)	1 (m)	1	0.1	3.5766×10^{-5}	802	4.4467×10^{-8}	802
none	1 (m)	all	0	1	3.9917×10^{-6}	1203	2.3454×10^{-6}	1203
1(m)	1 (m)	4	0.01	1	5.1192×10^{-6}	1203	1.1807×10^{-5}	1203
1(m)	1(m)	1 (e)	0.1	1	8.2154×10^{-7}	2008	1.0577×10^{-6}	802
1 (m)	1(m)	1 (m)	1	1	1.0218×10^{-6}	1203	1.5913×10^{-6}	1203
all	all	all	0.01	0.01	7.7030×10^{-7}	802	1.0189×10^{-6}	802
all	all	all	0.1	0.1	7.1193×10^{-6}	1203	1.3589×10^{-5}	802
all	all	all	1	1	8.2348×10^{-7}	2419	1.0675×10^{-4}	1604

Table 8: Perturbation locations (m=middle, e=end), Perturbation strength (deterministic/random), final error in ODE solver after fsolve, number of function evaluations (F.Eval.), Dirichlet BCs.

C.2 Table: Continuation Results in One Dimension

In this section the results for the continuation code in one dimension can be found in Table 10.

C.3 Table: Continuation Results in One Dimension

In this section the results for the continuation code in two dimension are presented in Table 11.

Perturbation Location			Perturbation		Deterministic Pert.		Random Pert.	
ρ	q	t	ρ	q	Error	F.Eval.	Error	F.Eval.
none	none	none	0	0	0	401	0	401
all	none	4	0.01	0	0.0014 (n) (1.8336×10^{-4})	1222	9.0622×10^{-8}	802
all	none	1 (e)	0.1	0	0.0158 (n) (0.0048)	822	3.1240×10^{-8}	802
all	none	1 (m)	1	0	4.0261×10^{-7}	1604	8.3985×10^{-8}	1203
none	4	all	0	0.01	1.2371×10^{-7}	802	1.4790×10^{-7}	802
4	4	4	0.01	0.01	2.3738×10^{-7}	802	1.8239×10^{-7}	802
4	4	1 (e)	0.1	0.01	3.2215×10^{-8}	802	1.2680×10^{-7}	802
4	4	1 (m)	1	0.01	7.5971×10^{-8}	1203	6.0599×10^{-6}	2010
none	1	all	0	0.1	2.0191×10^{-7}	802	2.4552×10^{-7}	802
1(e)	1(e)	4	0.01	0.1	2.2441×10^{-7}	802	2.9631×10^{-7}	802
1 (e)	1 (e)	1 (e)	0.1	0.1	9.4298×10^{-8}	802	8.5645×10^{-8}	802
1 (e)	1(e)	1 (m)	1	0.1	2.9226×10^{-6}	1203	9.1285×10^{-8}	802
none	1 (m)	all	0	1	5.7085×10^{-8}	1604	1.2191×10^{-7}	1203
1(m)	1 (m)	5	0.01	1	2.5055×10^{-6}	1203	2.8089×10^{-8}	1203
1(m)	1(m)	1 (e)	0.1	1	4.9564×10^{-8}	1203	9.8943×10^{-7}	802
1 (m)	1(m)	1 (m)	1	1	1.6979×10^{-7}	1203	3.9601×10^{-8}	1203
all	all	all	0.01	0.01	2.4752×10^{-5}	1605	5.7575×10^{-8}	802
all	all	all	0.1	0.1	1.7819×10^{-5}	2408	1.3962×10^{-6}	802
all	all	all	1	1	2.6975 (n) (1.3831×10^{-6})	2025	1.3221×10^{-7}	2005

Table 9: Perturbation locations (m=middle, e=end), Perturbation strength (deterministic/random), final error in ODE solver after fsolve, number of function evaluations (F.Eval.), Neumann BCs, n=no solution found (solutions found by increasing N can be seen in brackets).

$N = 20$			$N = 40$		
α	Funct. Evals.	First Order Opti.	α	Funct. Evals.	First Order Opti.
0.006	3609	4.6426×10^{-4}	0.0037	8812	0.0261
0.0142	3609	0.0015	0.0088	7209	0.0717
0.0277	3609	0.0126	0.0156	7209	0.0011
0.0490	4010	0.0034	0.0252	7209	0.0370
0.0539	2807	0.0217	0.0368	8010	0.0133
0.0567	2406	0.0985	0.0539	8010	0.0302
0.0596	2807	4.304×10^{-5}	0.0561	5607	0.0047
0.0626	2807	2.9463×10^{-4}	0.0584	5607	0.0017
0.0658	2807	2.6433×10^{-5}	0.0608	4806	0.0933
0.0692	2807	1.4200×10^{-4}	0.0632	5607	0.0091
0.0727	2807	6.5032×10^{-5}	0.0658	5607	0.0028
0.0764	2807	2.6970×10^{-4}	0.0685	5607	0.0066
0.0803	2807	7.2583×10^{-4}	0.0713	5607	0.0047
0.0844	2807	0.0229	0.0742	5607	0.0072
0.0896	2807	0.0055	0.0772	5607	0.0804
0.0951	3208	2.3774×10^{-4}	0.0803	5607	0.0025
0.1010	3208	3.3576×10^{-4}	0.0836	5607	0.0539
			0.0870	6408	0.0182
			0.0905	6408	0.0075
			0.0942	6408	0.0447
			0.0980	5607	0.0754
			0.1020	7210	0.0171

Table 10: Critical α values, number of function evaluations and first order optimality, for $N = 20$, $N = 40$ and $n = 11$.

α	Funct. Evals.	First Order Opti.
3.4523×10^{-4}	10005	0.0105
7.4002×10^{-4}	10005	0.0237
0.0012	10005	0.0377
0.0017	10005	0.0219
0.0023	10005	0.0206
0.0031	10005	0.0617
0.0037	10005	0.0293
0.0045	10005	0.0433
0.0055	10005	0.0796
0.0066	12006	0.0027
0.0080	12006	0.0289
0.0097	12006	0.0536
0.0117	14007	0.0236

Table 11: Critical α values, number of function evaluations and first order optimality, in two dimensions, for $N_1 = 10$, $N_2 = 10$ and $n = 11$.