Optimisation and Control Methodologies for Large-Scale and Multi-Scale Systems

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

Ioannis Bonis

School of Chemical Engineering and Analytical Science

2010
Contents

List of figures ................................................................................................................6
List of tables ..................................................................................................................8
Abstract ......................................................................................................................10
Declaration ..................................................................................................................12
Copyright statement .................................................................................................13
Acknowledgements ....................................................................................................14
Publications and presentations ..................................................................................15
Nonclemanture ..........................................................................................................18

1. Introduction ...........................................................................................................23
   1.1 Distributed parameter and multi-scale systems .................................................23
   1.2 Optimisation and control ..................................................................................25
   1.3 Equation-free model order reduction .................................................................29
   1.4 Motivation, objectives and overview of this work ..............................................31
   1.5 Organisation of the thesis ..................................................................................34

2. Literature review .....................................................................................................36
   2.1 Large-scale constrained optimisation methods ...................................................36
   2.2 Libraries for Nonlinear Programming ...............................................................44
   2.3 Inequality-constrained, large-scale optimisation ................................................46
   2.4 Control of distributed parameter and multi-scale systems ...............................49
   2.5 Model reduction methods for control systems ..................................................54
   2.6 Equation-free methods in optimisation and control ...........................................58
3. Model reduction-based PDE-constrained optimisation of large-scale dissipative systems

3.1 Introduction ................................................................. 68
3.2 Sequential Quadratic Programming ..................................... 70
3.3 The reduced Hessian SQP methods ................................... 73
3.4 Model reduction ............................................................. 77
   3.4.1 The reduction approach ............................................... 77
   3.4.2 The effect of model reduction in the optimisation ............. 81
3.5 The proposed optimisation algorithm .................................. 83
   3.5.1 Using an existing iterative solver for the computation of a feasible point .... 84
   3.5.2 Model reduction and computation of the reduced quantities .............. 85
   3.5.3 Computation of the reduced Hessian and the Lagrange multipliers .......... 87
   3.5.4 Updating the solution .................................................. 89
   3.5.5 Convergence monitoring and termination ........................... 89
3.6 A computationally efficient implementation .......................... 91
3.7 Case study: Optimisation of a tubular reactor ..................... 93
   3.7.1 The base case ......................................................... 93
   3.7.2 A 1-dof case ............................................................. 95
   3.7.3 A 3-dof case study ..................................................... 102
   3.7.4 Computational cost ..................................................... 106
3.8 Systems involving Dirichlet boundary conditions ................... 107
3.9 Summary ........................................................................ 110

4. Large-scale nonlinear programming of dissipative systems with nonlinear inequality constraints

4.1 Introduction ................................................................. 112
4.2 Large-scale inequality constrained nonlinear programming .................... 114
4.3 The notion of Partial Reduction in tackling with inequality constraints .......... 116
4.4. Handling inequalities with constraint aggregation functions ................. 119
4.5. Case study ................................................................... 121
4.6. Summary ...................................................................................................................127

5. “Equation-free” linear Model Predictive Control of nonlinear Distributed Parameter Systems ..........................................................130
  5.1 Introduction ...............................................................................................................130
  5.2 Model Predictive Control ........................................................................................132
  5.3. Model reduction ......................................................................................................137
  5.4. The proposed predictive control algorithm ..........................................................141
  5.5. A modification for the reduction of computational cost ........................................145
  5.6. Case studies ............................................................................................................148
    5.6.1. The base case: control of a tubular reactor ....................................................148
    5.6.2. Stabilisation of the reactor .............................................................................153
    5.6.3. Destabilisation of the reactor ........................................................................157
    5.6.4. Computational cost .........................................................................................161
  5.7. Summary ................................................................................................................161

6. Conclusions and recommendations for future work .............................................163
  6.1 Conclusions ..............................................................................................................163
  6.2 Future work ............................................................................................................167

References ....................................................................................................................170

Appendix: Source code ..................................................................................................192

  A.1 FORTRAN Codes corresponding to Chapter 3 .......................................................192
    A.1.1 1-dof case study, Algorithm 3.1 ......................................................................192
    A.1.2 3-dof case study, Algorithm 3.1 ......................................................................196
    A.1.3 1-dof case study, Algorithm 3.2 ......................................................................201
    A.1.4 3-dof case study, Algorithm 3.2 ......................................................................206
    A.1.5 Utilities: subroutines used in the codes of Sections A.1.1 – A.1.4 ..................212
  A.2 FORTRAN Codes corresponding to Chapter 4 .....................................................220
    A.2.1 3-dof case study optimised using Algorithm 4.1 (PRSQP) .........................220
    A.2.2 3-dof case study optimised using Algorithm 4.2 (KS) .................................229
A.3 MATLAB Codes corresponding to Chapter 5 .............................................................235
   A.3.1 Implementation of the control Algorithm 5.2.....................................................235
   A.3.2 Implementation of the control Algorithm 5.3.....................................................236
   A.3.3 Utilities: functions used by the previous codes ...................................................238
   A.3.4 Simulator.............................................................................................................240
   A.3.5 Script for plotting...............................................................................................244

Final word count: 52512
List of figures

Figure 3.1: Eigenspectrum of a dissipative system exhibiting a clear separation of modes. 78

Figure 3.2: Schematic representation of a tubular reactor ................................................... 93

Figure 3.3: Bifurcation diagram for the tubular reactor dimensionless variables: concentration x₁ and temperature x₂........................................................................................................ 96

Figure 3.4: The convergence paths for the proposed algorithm and a feasible point reduced Hessian method. ............................................................................................................................................... 97

Figure 3.5: The convergence curve for the 1 dof case study................................................. 97

Figure 3.6: The profiles for the dimensionless concentration and temperature for the optimal Da........................................................................................................................................................................... 99

Figure 3.7: Reduction of the time taken for the update of the basis for the dominant subspace per iteration for the 1-dof case study........................................................................................................................................................................ 101

Figure 3.8: The convergence curve for the 1 dof case study using Algorithm 3.2.............. 101

Figure 3.10: The profiles for the dimensionless concentration and temperature for the reactor and the cooling zones, for the optimal x₃w.................................................................................................................. 104

Figure 3.11: The convergence curve for the 3 dof case study using Algorithm 3.2............... 106

Figure 4.1: Schematic representation of a tubular reactor with 3 cooling zones ............ 122

Figure 4.2: The convergence curve using Algorithm 4.1 (PRSQP). .................................... 126

Figure 4.3: The convergence curve using Algorithm 4.2 (KS). ........................................ 127
Figure 4.4: Dimensionless concentration (a) and temperature (b) profiles at the optimum using Algorithms 1 and 2 plotted versus the base scenario without inequalities..............128

Figure 5.1: Eigenspectrum of a discretised dynamic system, exhibiting a clear separation of time-scales. ...........................................................................................................................139

Figure 5.2: Schematic representation of a tubular reactor with 8 cooling zones. ..........149

Figure 5.3: Dynamic profile for the dimensionless (a) temperature and (b) concentration for the open loop reactor behaviour with \( r = 0 \).................................................................................................151

Figure 5.4: Dynamic profile for the dimensionless temperature and concentration for the open loop reactor with \( r = 0.5 \). ........................................................................................................152

Figure 5.5: (a) Dynamic behaviour of the open-loop system, reference output and closed-loop output and (b) the corresponding computed cooling profiles. .........................154

Figure 5.6: Dynamic profiles for the dimensionless concentration and temperature throughout the reactor for the closed loop system. ...............................................................155

Figure 5.7: (a) Dynamic behaviour of the reference output and the closed loop output for the conditional model update case and (b) Corresponding computed cooling zones’ profiles ..........................................................................................................................157

Figure 5.8: (a) Closed loop output and reference for the destabilisation case study and (b). the corresponding calculated inputs .................................................................158

Figure 5.9: The closed loop dynamic profiles for the dimensionless concentration and dimensionless temperature throughout the reactor for the destabilisation case study. ...159

Figure 5.10: (a) The closed loop and reference output for the destabilisation case study using Algorithm 5.2. (b) Corresponding computed cooling profiles........................................160
List of tables

List of Latin symbols............................................................................................................... 18

List of Greek symbols............................................................................................................... 21

List of Abbreviations ................................................................................................................ 22

Table 3.1: Algorithm 3.1: The proposed optimisation algorithm ............................................ 90

Table 3.2: Algorithm 3.2: An efficient implementation or the proposed algorithm .......... 92

Table 3.3: Convergence data for the 1 dof case study ........................................................... 98

Table 3.4: Convergence data for the 1 dof case study using Algorithm 3.2 ..................... 100

Table 3.5: Convergence data for the 3dof case study using Algorithm 3.1 ....................... 103

Table 3.6: Convergence data for the 3dof case study using Algorithm 3.2 ....................... 105

Table 3.7: Computational cost (CPU seconds) of the optimisation procedure ............... 106

Table 4.1: Algorithm 4.1: the proposed optimisation scheme with inequalities being handled by partial reduction .................................................................................................................. 118

Table 4.2: Algorithm 4.2: the proposed optimisation scheme with inequalities being handled with a KS function .................................................................................................................. 120

Table 4.3: Results obtained with different strategies .......................................................... 124

Table 4.4: Convergence data using Algorithm 4.1............................................................ 124

Table 4.5: Convergence data using Algorithm 4.2............................................................ 125
Table 5.1: Algorithm 5.1: The adaptive linearisation scheme ............................................ 143

Table 5.2: Algorithm 5.2: The proposed control scheme.................................................... 145

Table 5.3: Algorithm 5.3: An efficient implementation of the control scheme............... 147

Table 5.4: Computational cost (CPU minutes) of the closed-loop simulations ............... 161
Abstract

Distributed parameter systems (DPS) comprise an important class of engineering systems ranging from “traditional” such as tubular reactors, to cutting edge processes such as nano-scale coatings (Christofides, 2001). DPS have been studied extensively and significant advances have been noted, enabling their accurate simulation. To this end a variety of tools have been developed. However, extending these advances for systems design is not a trivial task (Biegler and Wachter, 2003). Rigorous design and operation policies entail systematic procedures for optimisation and control. These tasks are “upper-level” and utilize existing models and simulators. The higher the accuracy of the underlying models, the more the design procedure benefits. However, employing such models in the context of conventional algorithms may lead to inefficient formulations.

The optimisation and control of DPS is a challenging task. These systems are typically discretised over a computational mesh, leading to large-scale problems. Handling the resulting large-scale systems may prove to be an intimidating task and requires special methodologies. Furthermore, it is often the case that the underlying physical phenomena span various temporal and spatial scales, thus complicating the analysis. Stiffness may also potentially be exhibited in the (nonlinear) models of such phenomena. The objective of this work is to design reliable and practical procedures for the optimisation and control of DPS.

It has been observed in many systems of engineering interest that although they are described by infinite-dimensional Partial Differential Equations (PDEs) resulting in large discretisation problems, their behaviour has a finite number of significant components (Christofides, 2001), as a result of their dissipative nature. This property has been exploited in various systematic model reduction techniques. Of key importance in this work is the identification of a low-dimensional dominant subspace for the system. This subspace is heuristically found to correspond to part of the eigenspectrum of the system and can therefore be identified efficiently using iterative matrix-free techniques. In this light, only low-dimensional Jacobians and Hessian matrices are involved in the formulation of the
proposed algorithms, which are projections of the original matrices onto appropriate low-dimensional subspaces, computed efficiently with directional perturbations.

The optimisation algorithm presented employs a 2-step projection scheme, firstly onto the dominant subspace of the system (corresponding to the right-most eigenvalues of the linearised system) and secondly onto the subspace of decision variables. This algorithm is inspired by reduced Hessian Sequential Quadratic Programming methods and therefore locates a local optimum of the nonlinear programming problem given by solving a sequence of reduced quadratic programming (QP) subproblems (Biegler et al., 1995). This optimisation algorithm is appropriate for systems with a relatively small number of decision variables.

Inequality constraints can be accommodated following a penalty-based strategy which aggregates all constraints using an appropriate function (Itle et al., 2004), or by employing a partial reduction technique in which only equality constraints are considered for the reduction and the inequalities are linearised and passed on to the QP subproblem (Schulz and Bock, 1997).

The control algorithm presented is based on the online adaptive construction of low-order linear models used in the context of a linear Model Predictive Control (MPC) algorithm (Camacho and Bordons, 2004), in which the discrete-time state-space model is recomputed at every sampling time in a receding horizon fashion. Successive linearisation around the current state on the closed-loop trajectory is combined with model reduction, resulting in an efficient procedure for the computation of reduced linearised models, projected onto the dominant subspace of the system. In this case, this subspace corresponds to the eigenvalues of largest magnitude of the discretised dynamical system. Control actions are computed from low-order QP problems solved efficiently online.

The optimisation and control algorithms presented may employ input/output simulators (such as commercial packages) extending their use to upper-level tasks. They are also suitable for systems governed by microscopic rules, the equations of which do not exist in closed form. Illustrative case studies are presented, based on tubular reactor models, which exhibit rich parametric behaviour.
Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Ioannis Bonis
Copyright statement

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://www.campus.manchester.ac.uk/medialibrary/policies/intellectual-property.pdf), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses.
Acknowledgements

First and foremost I need to thank my supervisor, Dr Kostas Theodoropoulos, for guiding me throughout this PhD study and for giving me the opportunity to embark on it in the first place.

I would also like to thank Michael and Weiguo for their friendship and help, as well as all the students of the Centre for Process Integration, past and present, whom I had the pleasure of meeting and working with.

I need to express my gratitude to the people who contributed to my remaining (quasi-)sane throughout this long process, sometimes by driving me insane: my sister, my parents, Stella, my friends from Greece and Manchester. At good and bad times they have been the pillar holding the structure of my life together.
Publications and presentations

A. Publications in journals

A1. Bonis I., Xie W., Theodoropoulos C., A linear model predictive control algorithm for nonlinear large-scale distributed parameter systems, AICHE Journal, accepted for publication

A2. Xie W., Bonis I., Theodoropoulos C., Off-line Model Reduction for On-line Linear MPC of Nonlinear Large-Scale Distributed Systems, Computers and Chemical Engineering, in press

A3. Bonis I., Theodoropoulos C., Model reduction-based steady-state optimization using large-scale steady-state simulators, submitted for publication

B. Publications in peer-reviewed proceedings volumes


B4. Bonis I., Valiño-Pazos S., Fragkopoulos I.S., Theodoropoulos C., Modelling of micro- and nano-patterned electrodes for the study and control of spillover processes in catalysis, Computer Aided Chemical Engineering, *to appear*

C. Publications in non-reviewed proceedings


D. Oral Presentations


D2. Bonis I., Theodoropoulos C., Model Reduction-Based Constrained Optimisation for Large-Scale Steady State Systems Using Black-Box Simulators, American Institute of Chemical Engineers (AICHE) Annual Meeting, Philadelphia, USA, 7-12 November 2008


D4. Bonis I., Theodoropoulos C., Steady-state constrained optimisation for input/output large-scale systems using model reduction technology, CICADA Workshop on Hybrid Systems and Model Reduction, Manchester, UK, 19-20 March 2009

D5. Bonis I., Theodoropoulos C., A reduced Sequential Quadratic Programming method for large systems exploiting existing simulators, Coping with Complexity: Model Reduction and Data Analysis workshop (part of the Algorithms for Approximation VI conference), Ambleside, Lake District, UK, 31 August – 4 September 2009

D7. Bonis I., Theodoropoulos C., Reduced Linear Model Predictive Control for Non-Linear Distributed Parameter Systems, American Institute of Chemical Engineers (AICHE) Annual Meeting, Philadelphia, USA, 7-12 November 2010
Nonclemanture

List of Latin symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$, $B$, $C$, $D$</td>
<td>Matrices involved in the state-space linear models (continuous time)</td>
</tr>
<tr>
<td>$A_d$, $B_d$, $C$, $D$</td>
<td>Matrices involved in the state-space linear models (discrete time)</td>
</tr>
<tr>
<td>$A_{c,\text{red}}$, $B_{c,\text{red}}$</td>
<td>Projections of $A$, $B$ onto the dominant subspace (continuous time)</td>
</tr>
<tr>
<td>$A$, $B$</td>
<td>Projections of $A_d$, $B_d$ onto the dominant subspace (discrete time)</td>
</tr>
<tr>
<td>$B$</td>
<td>The Hessian of the Lagrangian</td>
</tr>
<tr>
<td>$B_R$</td>
<td>The reduced Hessian of the Lagrangian</td>
</tr>
<tr>
<td>$C_m$</td>
<td>Controllability matrix</td>
</tr>
<tr>
<td>$d$</td>
<td>Search direction of the QP subproblem</td>
</tr>
<tr>
<td>$dof$</td>
<td>Number of the independent variables</td>
</tr>
<tr>
<td>$f$</td>
<td>The objective function</td>
</tr>
<tr>
<td>$f_{KS}$</td>
<td>Merit function incorporating the aggregated inequality constraints</td>
</tr>
<tr>
<td>$F(x,u,t)$</td>
<td>Fixed point procedure which models the dynamical system</td>
</tr>
<tr>
<td>$F_x$</td>
<td>The Jacobian computed using the time integrator</td>
</tr>
<tr>
<td>$G(x)$</td>
<td>The equality constraints for the optimisation problem</td>
</tr>
<tr>
<td>$h(x)$</td>
<td>The inequality constraints for the optimisation problem</td>
</tr>
</tbody>
</table>
$H$  The reduced Jacobian  

$H(.)$  Heaviside function  

$J$  Jacobian of $G$  

$KS(.)$  The Kresisselmeier – Steinhauser function  

$L(.)$  Lagrange function  

$m$  Number of dominant modes  

$N$  Number of dependent (state) variables  

$N_{in}$  Number of inequality constraints  

$N_p$  Prediction horizon for the MPC problem  

$N_c$  Control horizon for the MPC problem  

$n_{in}$  Number of inputs  

$n_{out}$  Number of outputs  

$O_m$  Observability matrix  

$P$  The dominant subspace of the system  

$P_{ext}$  Extended dominant subspace of the system including the space of independent variables  

$Q$  The complement subspace of $P$  

$p_y$  Y-component of the search direction  

$p_z$  Z-component of the search direction  

$t$  Time
$T_s$  
Sampling time

$J$  
The Jacobian of the system

$\tilde{x}$  
Scaled variable vector

$\hat{x}$  
Estimate of $x$

$\hat{x}_{\text{nom}}$  
Nominal state variable

$\hat{y}$  
Model-based predictions for future outputs

$\tilde{u}_{\text{nom}}$  
Nominal input variable

$Z$  
Matrix containing future inputs calculated from the MPC controller

$u$  
The dependent variable vector

$x$  
The variable vector: $x^T = [u^T \ z^T]$ for optimisation applications. The state variables vector for control.

$Y$  
Basis for the complement space of the independent variables

$Z$  
A basis for the space of the independent variables

$Z^*$  
The counterpart of $Z$ computed using a reduced model

$z$  
The independent variable vector (the system input for control)

$\hat{Z}_{\text{ext}}$  
Extended basis of the dominant subspace to include decision variables

$\hat{Z}$  
An orthonormal basis for the dominant subspace of the system

$Z_r$  
A basis for the subspace of the independent variables
### List of Greek symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Steplength for the line search</td>
</tr>
<tr>
<td>$\xi$</td>
<td>Reduced state vector</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Lagrange multipliers</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Penalty parameter</td>
</tr>
<tr>
<td>$\mu_i, i=1,N$</td>
<td>Eigenvalue</td>
</tr>
<tr>
<td>$\nu$</td>
<td>Reduced dependent variable vector</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Reduced Lagrange multipliers</td>
</tr>
<tr>
<td>$\chi$</td>
<td>Reduced state variable vector</td>
</tr>
<tr>
<td>$\varphi_{\mu}$</td>
<td>Merit function</td>
</tr>
</tbody>
</table>
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS</td>
<td>Distributed Parameter System</td>
</tr>
<tr>
<td>KKT</td>
<td>Karush – Kuhn – Tucker</td>
</tr>
<tr>
<td>KS</td>
<td>Kresisselmeier – Steinhauser</td>
</tr>
<tr>
<td>LPS</td>
<td>Lumped Parameter System</td>
</tr>
<tr>
<td>MOR</td>
<td>Model Order Reduction</td>
</tr>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>NLP</td>
<td>Nonlinear Programming</td>
</tr>
<tr>
<td>NMPC</td>
<td>Nonlinear Model Predictive Control</td>
</tr>
<tr>
<td>NAND</td>
<td>Nested Analysis and Design</td>
</tr>
<tr>
<td>PDE</td>
<td>Partial Differential Equation</td>
</tr>
<tr>
<td>POD</td>
<td>Proper Orthogonal Decomposition (also termed Karhunen Loève expansion and Principal Component Analysis)</td>
</tr>
<tr>
<td>RPRH</td>
<td>Recursive Projective Reduced Hessian</td>
</tr>
<tr>
<td>rSQP</td>
<td>Reduced-space Sequential Quadratic Programming</td>
</tr>
<tr>
<td>PRSQP</td>
<td>Partially Reduced Sequential Quadratic Programming</td>
</tr>
<tr>
<td>SAND</td>
<td>Simultaneous Analysis and Design</td>
</tr>
<tr>
<td>SL</td>
<td>Successive Linearisation</td>
</tr>
<tr>
<td>SQP</td>
<td>Sequential (or Successive) Quadratic Programming</td>
</tr>
<tr>
<td>QP</td>
<td>Quadratic Programming</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Distributed parameter and multi-scale systems

A system is a collection of components governed by specific rules and possibly exhibits characteristics that its components do not. Systems which involve variables exhibiting spatial variations are termed Distributed Parameter Systems (DPS) (Ahmed and Robert, 2001). Thus, the time evolution of DPS can only be described in the infinite-dimensional state space. Contrary to DPS, lumped parameter systems (LPS) can be adequately modelled using variables, which do not exhibit significant spatial variation. Hence such variations are neglected in LPS, which can be described with a finite number of variables. An example of such systems is the continuous stirred tank reactor (CSTR) for which the assumption of uniform concentration holds. If mixing phenomena cannot be neglected, then a DPS model needs to be formulated. Both classes of systems arise naturally in engineering practice, with the handling of DPS being more challenging mainly due to the computational costs associated with it.

Most engineering DPS can be accurately modelled using sets of integro-differential equations describing changes along the spatial and temporal dimensions. Momentum, mass, heat and electrical charge transfer are among the phenomena, which can be described adequately using continuum equations. Typically more than one phenomenon takes place in systems of engineering interest; hence the corresponding models consist of sets of equations, which need to be solved simultaneously. These equations are typically nonlinear, and are solved computationally rather than analytically. In particular, a computational mesh is defined on the domain of the physical system considered, and it is on it that the equations are solved. Usually, a local set of equations is defined for each
node of the discretised model, so the total number of variables involved in the model is typically large. Large-scale systems are very common in all fields of engineering. The size of the resulting system depends on the discretisation method employed, the number of nodes (or elements) considered (depending on the nonlinearity of the equations), as well as on the geometrical aspects of the physical system. Thankfully, the discretisation systems only involve variables defined in the neighbouring nodes. This leads to sparse systems, the structure of which can be exploited in efficient simulation schemes.

For the simulation of practically all classes of DPS used in production systems, various simulators have been developed. These vary from commercial to home-made codes, but in any case they are assumed to comprise reliable approximations of the physical systems. More often than not, the user has limited access to the commercial packages. The latter can be used to set up custom models on one hand, but the information which can be retrieved is limited and predefined by the company that produces the software. Even for in-house developed codes, the user does not necessarily have access to the source code for various possible reasons, or even if the access is granted, modifying the code could be an overwhelming task. It is always desirable to build on existing information and avoid “reinventing the wheel” by ignoring already existing solutions. Thus, it would be advantageous to develop algorithms to perform system-level tasks (in this case for optimisation and control), which build on existing simulators, extending their use. In some cases, it is not realistic to neglect previous work performed, as it corresponds to man-years of work (e.g. in the case of commercial Computational Fluid Dynamics packages or Process Simulators). Those codes are not always capable of performing “upper-level” tasks such as optimisation and control (e.g. the very popular CFD packages CFX* and OpenFoam* do not offer optimisation options). Or even if they do include such options perhaps the algorithms they employ or the limitations they impose are considered very restricting. For such cases, efficient algorithms need to be developed, which both utilize existing simulators, exploiting the advantages this incurs and waiving the restricting limitations they impose.

An important property many engineering systems exhibit is dissipation, i.e. conversion of useful work to heat due to forces such as friction (Willems, 1972; Brogliato et al., 2007). Production of heat leads to an increase of temperature (and entropy) as well as reduction of energy. Many systems of interest to the Process Systems Engineering community are dissipative (Alonso and Ydstie, 1996; Christofides, 2001). This property leads to dynamic behaviour which exhibits characteristics one may exploit for analysis.
1. Introduction

Systems which span various temporal and spatial scales are termed multi-scale (Kevrekidis et al., 2003a). Hence, the term refers to the characteristics of the physical system rather than to the simulation approach. Multi-scale systems can be classified as DPS or LPS. A dominant trend in multi-scale systems modelling is the coupling of appropriate simulators for the various length-scales, which are significant for the system. Coupling micro-, meso- and/or macro-scopic simulators, is not a trivial task but it constitutes an intuitively plausible approach, which results in simulators of high accuracy. An indicative application of this approach, in heterogeneous catalytic systems for example, is the use of a macroscopic model for the simulation of the momentum and mass transport in the bulk of the fluid, and of a microscopic model for the surface diffusion and reaction on the catalyst (Kapil et al., 2010). The coupled models need not be similar, for example the (deterministic) Navier-Stokes equations for momentum transport can be coupled with a stochastic lattice kinetic Monte Carlo model for the surface phenomena on the catalyst.

Not all multi-scale systems are simulated using hybrid simulators. For example in fluid flow in a pipe, the boundary layer formed in the vicinity of the pipe walls is characterized with a smaller length-scale as the one of the bulk fluid flow (Schlichting et al., 2000). Another example comes from the modelling of turbulence: although such systems are typically modelled using Partial Differential Equations, they are multi-scale, a fact which is often exploited in computations (Hsu et al., 2010).

In this work, the focus is on multi-scale systems in general. The existence of various time-scales is being used for extracting a reduced model using an existing simulator in an input/output fashion. The case studies come from continuum models, however since no restrictive assumptions are made, the use of any multi-scale simulator would be suitable within the proposed framework.

1.2 Optimisation and control

Optimisation and control methods constitute “upper-level” tasks (Biegler and Wachter, 2003), which presuppose a description of the system at hand. Optimisation is performed at the design stage of an industrial system and before any retrofitting is implemented (Aguilar
et al., 2007). In addition, real-time optimisation is used for the computation of process operation conditions, subject to variations on the feed, the desired product specifications or ambient conditions. It plays an invaluable role in the design and operation of sustainable industrial systems (Diehl et al., 2002).

The objectives of optimisation differ significantly between applications. Typically the objective is of economic nature: a cost is minimised or a profit is maximized, with respect to values of specific independent variables (also called degrees of freedom and decision variables), subject to the satisfaction of a model for the system, possible physical, manufacturing or environmental limitations and bounds for the decision variables. The identification of a meaningful objective and the selection of realistic decision variables and corresponding bounds is imperative for the success of the venture (Biegler et al., 1997; Edgar et al., 2001).

Optimisation problems can be classified as linear or nonlinear depending on the nature of the objective function and the constraints, constrained or unconstrained, integer, discrete or continuous depending on the classes of variables they involve, large, medium or small depending on the number of variables it contains and so forth (Nocedal and Wright, 1999; Edgar et al., 2001; Biegler and Grossmann, 2004). Roughly speaking, NLP problems are considered small if they involve less than 40 variables, medium if they involve less than 200 and large if the number of variables in the formulation is greater than 200 (SAS, 2010). Depending on the optimisation problem and the objective, an optimisation method is selected. There does not exist a single optimisation methodology suitable for all classes of problems. Each method is applicable to a specific class of problems and has its own strengths and limitations.

Optimisation techniques comprise a valuable decision tool for all specialties of engineers to employ in many of their activities. Typical applications in Process Engineering include synthesis and design of heat exchanger networks (Tantimuratha et al., 2001), separation units (Gadalla et al., 2005), chemical reactors (Asteasuain and Brandolin, 2008) and other pieces of apparatuses. Optimisation is also applied in process operations, such as scheduling (Maravelias and Grossmann, 2001) and supply chain management (Papageorgiou et al., 2001). Finally, an important class of applications for optimisation algorithms is control.
As far as nonlinear continuous optimisation is concerned, perhaps the most widely used optimisation strategy is Sequential Quadratic Programming (SQP). SQP solves the Nonlinear Programming (NLP) problem by applying Newton’s method to the first order Karush-Kuhn-Tucker (KKT) optimality conditions (Nocedal and Wright, 1999; Edgar et al., 2001). Many variants of SQP have been proposed, depending on the way the inequality constraints are handled, if the optimisation problem is solved simultaneously with the simulation one (hence if the optimisation path consists of feasible or infeasible points), on how the quadratic subproblem is solved, on how gradients are calculated, on which (and if) globalisation strategies are employed an so forth (Boggs and Tolle, 1995). SQP-based optimisers have been shown to involve the least number of function evaluations (Schittkowski, 1987).

The optimisation of large systems is not a trivial task and typically requires the utilization of special methods. To this end, the reduced Hessian SQP framework has been developed, where the sequence of subproblems solved is projected onto the null space of the equality constraints. This approach reduces the dimension of the subproblem and is appropriate for systems involving less decision than dependent variables (typically at least 1-2 orders of magnitude less than the dependent ones) (Biegler et al., 1995). Another approach to the optimisation of a given large-scale system is to simplify the model itself, so that conventional optimisation methods can be employed. A surrogate model or a systematically identified using model order reduction model can be used (Armaou and Christofides, 2002; Gadalla et al., 2003). The case of multi-scale systems is even more challenging, as in addition to their high dimensionality, they may exhibit additional issues such as noisy derivatives or unavailability of the governing equations in closed form (Bindal et al., 2006). The same remedies (use of surrogate models or model reduction) have been successfully applied for process control (Shvartsman and Kevrekidis, 1998; Aggelogiannaki and Sarimveis, 2008).

Optimisation and control are synergic. Optimisation typically provides specifications for the operating conditions, which are used as set-points for the controllers. If real-time optimisation is employed, updating of the set-points is performed online (Zanin et al., 2000; Roffel and Betlem, 2004). This description implies a sequential relation between optimisation and control, which might not be indicative of engineering practice. Indeed, most advanced control strategies employ optimisation algorithms for the calculation of the control actions (Zavala and Biegler, 2009a). Besides, control algorithms involving economic
objectives have appeared in the literature (Diehl et al., 2002; Zanin, 2002; Kadam and Marquardt, 2007). Thus, the border between optimisation and control algorithms in some cases is difficult, if not impossible, to define.

Control problems can be classified as tracking, disturbance rejection or model matching (Basar and Bernhard, 1995). Control applications include process systems (among those: reactors (Zavala and Biegler, 2009a), fuel cells (Hasikos et al., 2009), distillation columns (Skogestad, 1997), gas turbines (Ghorbani et al., 2008) and so forth), robotics, transportation (ships, trains, vehicles, trains), biomedical systems and so forth (Shinners, 1998). Most industrial systems incorporate controllers.

Research in the field of control can be traced as early as the 1930s (Bennett, 1993). One of the earliest attempts, and even now perhaps the most commonly used techniques is PID control (Proportional-Integral-Derivative controller), which employs linear single-input single-output (SISO) models and its design is based on a (Laplace) transformation to the frequency domain (Stephanopoulos, 1984). More modern control strategies include the predictive, nonlinear, adaptive and robust control (Shinners, 1998). Each technology has a specific class of problems it is applicable to and exhibits its own strengths and weaknesses.

Perhaps the most widely used advanced control strategy is Model Predictive Control (Richelet et al., 1978; García et al., 1989; Qin and Badgwell, 1997; Morari and Lee, 1999; Mayne et al., 2000). It is generic, in the sense that it combines elements from optimal, stochastic, multivariate control as well as control with dead time (Camacho and Bordons, 2004). One of its most useful features is the ability to handle constraints. Rather than a single control algorithm, MPC can more accurately be described as a collection of such. Some of the linear MPC strategies are: Dynamic Matrix Control, Model Algorithmic Control, Predictive functional Control, Extended Prediction Self Adaptive Control, Extended Horizon Adaptive Control (Camacho and Bordons, 2004; Roffel and Betlem, 2004).

Conventional MPC employs a linear model for the prediction of the future outputs from a process or plant subject to given inputs, within a quadratic optimisation problem, subject to linear constraints. The control actions are computed as the solution of the QP in question. A typical objective is the minimisation of the control energy required and at the same time the minimisation of the deviation from the set-point (Camacho and Bordons, 2004). The use of linear models can be restricting in some cases, since most engineering systems are nonlinear (Henson, 1998). A linear model cannot accurately approximate them
in large regions of operation. On the other hand using nonlinear MPC incurs additional issues that need be tackled. For example the optimisation problem may be non-convex and consequently exhibit multiple extrema or even be infeasible. Moreover, the computational cost is increased in comparison to cost of the linear MPC; this increase may be significant especially since a contemporary trend is to implement MPC on low-cost hardware (Roldao-Lopes et al., 2009).

Successive Linearisation MPC has the advantage of effectively handling highly nonlinear systems and simultaneously producing QP optimisation problems; thus not only the solution of the latter is unique, but also the whole procedure is typically of reduced cost in comparison to the NMPC (Li and Biegler, 1989). Model reduction has also been proposed as a remedy for the increased computational cost of NMPC (Dubljevic and Christofides, 2006).

1.3 Equation-free model order reduction

Model reduction techniques provide a systematic way of using a system to extract a derivative one, which exhibits some desired properties (Schilders et al., 2008). Another way of viewing this procedure is that the full system is transformed to another one, which is reduced by the undesirable properties of the initial one; this viewpoint perhaps being intuitively more compatible with the term model reduction. There are various objectives that can be accommodated with the model reduction framework. Typically, model reduction identifies a system, which can be simulated at a lower computational cost, involves a smaller number of variables and equations compared to the full systems (Schilders et al., 2008). However, model reduction methods do not always result in systems which are always simpler, of lower computational cost and of less accuracy compared to the full ones; the goal of reduction sometimes is different, for example the derivation of a system which is alleviated of the stiffness of the original model (Hadjinicolaou and Goussis, 1998), or a system which can be used in order to perform system level tasks, such as bifurcation analysis (Koronaki et al., 2003), optimisation (Luna-Ortiz and Theodoropoulos, 2005) and control (Christofides, 2001), for which the original model is not suitable. Moreover, some model reduction methodologies can be used to provide insight on the
physical system examined, by identifying the significant variables, which exhibit higher impact on the system (Kourdis et al., 2010).

Model reduction can either be physical (also termed operational) or mathematical (Schilders et al., 2008; Theodoropoulos, 2011). Physical reduction is the application of physical, chemical, or other insights for the simplification of a given model. In this work, the focus is on mathematical reduction methods, which employ formal mathematical reasoning to achieve the desired reduction. Many of those methods employ some form of projection onto subspaces which are of lower dimension than that of the original system, but can be used for the production of reduced models, which capture most of the information included in the full system (Christofides, 2001). The identification of those subspaces may be performed online or offline, depending on the methodology chosen. Among the popular model reduction techniques one can classify Proper Orthogonal Decomposition (POD) (Ly and Tran), Approximate Inertial Manifold (AIM) (Jones et al., 1995), Computational Singular Perturbations (CSP) (Hadjinicolaou and Goussis, 1998), Balanced Truncation (Stykel, 2006) and the equation-free methods (Kevrekidis et al., 2004).

Equation-free methods have been developed by Kevrekidis and co-workers, having been introduced by Theodoropoulos, Qin and Kevrekidis (2000). They are inspired by the Recursive Projection Method (RPM) (Shroff and Keller, 1993). RPM provides a computational superstructure (“wrapper”) for stabilizing and accelerating the convergence of black-box simulators. The unstable modes which would decelerate the convergence of the simulator, or result in the simulation failing altogether, are identified and Newton’s method is applied to the projection of the system onto this unstable subspace, to ensure convergence whereas the given simulator is used for performing Picard iterations on the complement (stable) subspace, so that the procedure results in a Newton-Picard scheme which can be used for continuation of (stable or unstable) steady-states. By-products of this process are the identification of a low-dimensional dominant subspace of the system, (which is the maximal invariant subspace corresponding to the unstable modes of the simulation scheme) and reduced quantities, such as the reduced Jacobian, which are projections of the original ones onto the subspace examined. The computation of such quantities in an input/output fashion is otherwise non-trivial. In the equation-free methods, this approach to black-box simulators is applied to microscopic or multi-scale simulators, cases for which there is no option of using a white-box technique, as equations
describing the corresponding phenomena in terms of macroscopic (system level, coarse) variables are unavailable although intuitively existent (Kevrekidis et al., 2003a).

The approach of retrieving a model from microscopic systems has given rise to various methods, which belong to the equation-free framework, such as the micro-Galerkin method (Kevrekidis et al., 2003a), the gap-tooth projective integration scheme (Kevrekidis et al., 2003a), patch dynamics (Kevrekidis et al., 2004) and so forth. The common denominator of those methods, which will be revisited and explained in the next Chapter, is a lifting step in which the system-level variables are used to produce a set of feasible initial conditions for the microscopic variables, an evolving step which consists on using the microscopic simulator for integrating the system starting from this initial conditions and a restricting step, which is the coarse graining of the microscopic variables. This framework enables multi-scale systems to be effectively used for various tasks such as simulation, optimisation and control, without using (unavailable) closures of the equations, but instead exploiting given simulators for short time system integration (“bursts”).

1.4 Motivation, objectives and overview of this work

Technological advance has led to the widespread use of more complex systems both in industry and in everyday life. This renders the design, analysis and operation of complex systems a crucial task. This is both because the systems of small spatial scales need to be manufactured in great detail to function properly and because they are more fragile, in that relatively small stresses or disturbances may be significant. Even “traditional” engineering systems, that have been used in production systems for many years, have a lot to benefit from the utilization of advanced computational tools. Products of higher purities can be produced from a given system, whereas the operation and dynamical behaviour can be more accurately predicted and controlled; the latter having a positive effect on safety, plant economics, abiding by environmental regulations and so forth.

In the previous sections of this chapter, the problem of optimisation and control of distributed parameter systems has been outlined. This problem is non-trivial, especially if the system is considered to involve multiple scales. Several methods have been proposed,
focusing on different classes of systems, entailing different assumptions and following different approaches. Some of those methods will be delineated in the next chapter. The objectives and the scope of this work are outlined below. The advance of computers science and the inexpensive and easy access to computational resources have not eliminated the need for computationally efficient algorithms for simulation, optimisation and control.

The objectives of this work are:

- to formulate an efficient framework for optimisation and control of dissipative large-scale systems,
- to extend the use of existing simulators to perform optimisation and control tasks
- to enable the use of accurate, rigorous models in optimisation and control
- to minimise the computational cost of the framework in question by employing mathematical model reduction techniques
- and finally to be able to solve equality and inequality constrained problems

The main idea on which this work relies is exploiting a low-dimensional, dominant subspace for model reduction. This subspace is heuristically identified as the invariant subspace belonging to certain modes, i.e. a part of the eigenspectrum of the underlying system. For a steady-state system, this is the subspace belonging to the eigenvalues of largest real part, whereas for a dynamical system, this space belongs to the eigenvalues of largest magnitude corresponding to the slowest time-scales. This dominant subspace is typically low-dimensional for many dissipative systems. The approximation of a full system with a low-dimensional projection exploiting modal decomposition is what this work is based on. This approach, of replacing the system by adaptively projecting onto the low-dimensional dominant subspace, is exploited in order to perform optimisation and control tasks. This work is concerned with deterministic, large-scale, continuous PDE-constrained optimisation and predictive control.

The optimisation methods developed focus on large-scale NLP problems, which involve a small number of decision variables, which is typically the case in design problems. They combine the decomposition approach outlined with the reduced Hessian Sequential Quadratic Programming method for this matter, resulting in a two-step projection scheme, first onto the dominant subspace of the system and then onto the subspace of the decision
variables. In this formulation, only low-dimensional Jacobians, Hessians and gradients are involved, which are projections of the original ones, computed efficiently with matrix-free methods in conjunction with directional perturbations.

This framework is appropriate for PDE-constrained optimisation. Handling of inequality constraints is enabled using a special conservative penalty function termed KS function, which aggregates all inequality constraints. Alternatively, reducing the problem only onto the null space of the equality constraints is considered in a partially reduced SQP framework, which passes the linearised inequalities to the underlyingQP subproblem.

The same approach for reduction is used in the context of model predictive control. In specific, the dominant subspace of the system at hand is identified on the current state (i.e. on the closed loop trajectory), and a low-order linearised system belonging to this subspace is produced, following a successive linearisation scheme. The linear model is used in the formulation of a linear MP Controller. This is an online procedure, which results in the design of controllers in a receding horizon fashion. A sequence of control actions are calculated from a reduced QP subproblem, the first of which is implemented at the next sampling time.

A significant feature of the methods developed, is that they are able to utilize existing simulators. Scientists and especially engineers typically prefer to use ready-made tools in their work rather than develop their own, given a choice. This does not imply a sluggish way of functioning, rather recognition that it would be fruitless to “reinvent the wheel” by implementing something, for which an implementation is already available and can be customised for the case at hand. It is often the case that the available tools are fruit of many person-years of work and consequently exhibit excellent levels of accuracy and efficiency. As an example the Computational Fluid Dynamics software is mentioned, which typically consists of tenths of thousands of lines of code and sometimes implements heuristics, which apply to classes of stiff problems. Moreover, developing custom tools is a tedious task and requires effort, which could in principle be directed to other tasks, that will extend, rather than overlap with previous work. Furthermore, for some classes of problems, the use of available packages is indispensable as those integrate a large number of data (statistical, experimental or other) which is hard to obtain from literature. This is the case for process (flowsheet) simulators such as Aspen or simulation tools such as
CHEMKIN. Hence, the use of techniques, which exploit already available knowledge, in this context in terms of available software, is advantageous.

1.5 Organisation of the thesis

This thesis is organized in six chapters. Chapter 2 provides a review of optimisation and control methods for distributed parameter and multi-scale systems, as well as applications. One can note three trends: devising and utilizing novel optimisation and control methodologies, which are able to handle those classes of systems, using model reduction as a tool which enables the use of accurate reduced models that were formulated in a systematic way using operational or mathematical model reduction within a framework of an already established strategy, or using surrogate models within such a framework.

In Chapter 3 an optimisation algorithm for large-scale systems spanning various temporal scales is presented. It is based on the reduced Hessian SQP method and hence is appropriate for equality-constrained optimisation problems involving less decision than dependent variables. As per the conventional rSQP scheme, the quadratic optimisation subproblem solved in each iteration is projected onto the null space of the equality constraints. However, in contrast to the rSQP, the computation of a basis for that subspace does not involve calculating and inverting the full Jacobian; rather a reduced one is used. A model reduction step precedes, in which a dominant subspace of the system is identified. This procedure leads to a two-step projection scheme, the first projection being onto the dominant subspace of the system and the second onto the subspace of independent variables.

Chapter 4 extends the optimisation method presented in Chapter 3, to problems involving nonlinear inequality constraints. To this end two approaches are followed; the first is based on a special conservative penalty function termed constraint aggregation function, which replaces all inequalities by a single conservative one, later incorporated to the merit function. The second approach followed has an advantage over the first in that it does not introduce additional parameters to the procedure. As in the equality-constrained version of the algorithm, a basis for the dominant subspace of the equalities is identified and used for
the two projections. The QP subproblem has additional inequality constraints, which stem from the linearisation of the nonlinear inequalities onto the subspace considered.

In Chapter 5 a model reduction-based model predictive control algorithm is presented. It exploits the separation of time-scales for model reduction, which is translated to the clustering of eigenvalues in the eigenspectrum. Eigenvalues of high magnitude correspond to slow dynamics; a subspace for those dynamics is adaptively identified and used for control. Fast modes are neglected but due to the adaptive nature of the technique they are captured at run-time if they become significant. The nonlinear model is linearised at every sampling interval and the reduced local linearised model is used for control purposes.

Finally, in Chapter 6 the conclusions of this work are outlined and recommendations for future work in the field of optimisation and control of large-scale and multi-scale systems are given.
2. Literature review

2.1 Large-scale constrained optimisation methods

In recent years, the issue of optimisation of large scale systems in general and most importantly PDE-constraint optimisation has received much attention (Biegler, 2003; Biegler, 2007). The general formulation of an optimisation problem is

\[
\min f(x) \\
\text{s.t. } G(x) = 0 \text{ and } \ x^l \leq x \leq x^u
\]  

(2.1)

where \( f: \mathbb{R}^{N_{\text{dof}}} \rightarrow \mathbb{R} \) is the objective function, \( G: \mathbb{R}^{N_{\text{dof}}} \rightarrow \mathbb{R}^N \) is the vector function of equality constraints (resulting from the discretisation of the set of PDEs) and \( x \in \mathbb{R}^{N_{\text{dof}}} \) is the joint vector of the dependent \( u \in \mathbb{R}^N \) and independent variables \( z \in \mathbb{R}^{dof} : x^T = [u^T \ z]^T \).

The optimisation problem (2.1) can be solved using stochastic or deterministic nonlinear programming methods. Stochastic methods have more chances of converging to the global optimum than deterministic ones and they have less requirements as to the smoothness of the function but on the other hand provide no guarantees for convergence to an optimum (Lundy and Mees, 1986; Kall et al., 1994). In this work, the focus is on deterministic methods. Perhaps the most widely used deterministic optimisation method for smooth nonlinear programming is Sequential Quadratic Programming (SQP) (Nocedal and Wright, 1999). SQP solves optimisation problems by iteratively solving the Karush-Kuhn-Tucker (KKT) first-order optimality conditions with the quasi-Newton’s method. At each iteration, a quadratic subproblem (QP) is solved. Within this framework both equality and inequality constraints can be accommodated and satisfied.
SQP is a powerful method for nonlinear programming problems with significant nonlinearities (Nocedal and Wright, 1999). A very successful and well formulated class of algorithms for large NLP based on SQP is the Reduced Hessian methods (Nocedal and Overton, 1985; Byrd and Nocedal, 1990; Biegler et al., 1993; Cervantes and Biegler, 1998; Liu, 2009). Reduced Hessian methods are also termed reduced space SQP (rSQP) methods. They are quasi-Newton methods and they can be considered as an extension of the work of Coleman and Conn (1984). Biegler and collaborators have made important contributions to this class of optimisation algorithms (Biegler et al., 1995; Biegler et al., 2000b).

Reduced Hessian methods are meant for equality constrained, large-scale optimisation problems with a relatively small number of degrees of freedom, the second derivatives of which are expensive to compute (Nocedal and Wright, 1999). They are based on the partitioning of the space into two subspaces, one of which is the subspace of the search direction of the standard SQP method. This way the underlying QP subproblem is reduced, as the objective function gradient and the Hessian of the Lagrangian can be projected onto that subspace. Thus, the subproblem can be expressed merely in terms of the (few) degrees of freedom.

There are quite a few variations of the original algorithm, which differ in the way the bases for the tangent space of the constraints are computed, the Hessian of the system is updated, the QP subproblem is solved, etc. Since their first introduction, reduced Hessian methods have been extensively studied. They have been applied to many problems of engineering interest. Indicative examples include process optimisation (e.g. Itle et al., 2004), shape optimisation (e.g. Hazra, 2007) and control (e.g. Albuquerque et al., 1997; Bartlett and Biegler, 2006).

Handling of inequality constraints within the reduced Hessian framework is a topic on which some work has been done. The algorithms and applications will be outlined in Section 2.3.

Ternet and Biegler (1998) have presented a series of enhancements to the original algorithm with the purpose of augmenting robustness, numerical stability (especially for large problems) and efficiency while simultaneously preserving the desirable convergence properties of the original method. They employ a version of the QPKWIK algorithm for the solution of the QP subproblem (Schmid and Biegler, 1994), modified so that the constraints for the trust region are dealt with more efficiently. This moves the search direction closer
2. Literature review

to the steepest decent one and further from the Newton step direction. The trust region is combined with line searches. The authors also present a way of updating the basis for the subspace of the independent variables without explicit calculation. For reasons of efficiency, they also consider solving the QP subproblem at every iteration only partially. They conclude their contribution by presenting numerical results from case studies which illustrate the behaviour of each one of the modifications to the original algorithm proposed.

Albuquerque et al. have developed an interior point SQP strategy for handling inequality constraints (Albuquerque et al., 1999). This approach is based on the reduced Hessian method to enable handling of large-scale optimisation problems. They consider transforming the nonlinear programming (NLP) problem to an equivalent, equality constrained NLP, which includes just one degree of freedom and can be solved following the reduced Hessian approach. This framework combines interior point methods with reduced SQP methods and is called ISQP. Interior point methods are known to be efficient in cases of constrained optimisation, which includes many inequality constraints and in this case such a method is applied for the solution of the QP subproblems that arise in the solution of large NLPs. The derived method has also desirable global convergence properties as it is based on rSQP. The authors show that the algorithm exhibits quadratic convergence since the reduced Hessian is asymptotically exact, a convergence rate which can be enhanced even more by including a second order information. They illustrate the efficiency of the proposed scheme by applying it for cases of optimal control and model predictive control.

Biegler and Wächter have extended the application of reduced space SQP methods for optimisation that includes both equality and inequality constraints using existing modelling systems (Biegler and Wachter, 2003). An important aspect of this work is that the use of input/output simulators is enabled through the use of tailored Simultaneous Analysis and Design (SAND) methods and matrix-free Krylov methods, the latter also enhancing the computational efficiency of the optimisation scheme. The authors consider both the direct tailored approach and the adjoint tailored approach. The second results in better performance, however the former employs a preconditioned Krylov method for Jacobian-vector products, which can be calculated even when the Jacobian itself is not explicitly provided to the optimiser. Second order information is used for updating the reduced Hessian method by employing a preconditioning matrix for the approximation of the
inverse of the projection of the system’s Hessian. This results in significant performance enhancement. However this approach renders an active set strategy for handling inequality constraints impractical. Thus a barrier approach is adopted, according to which a logarithmic barrier term is added to the objective function. The barrier parameters are decreasing as the procedure converges to the optimal point. For the solution of this barrier problem, for a given barrier parameter, a primal-dual approach is followed, which provides search directions both for the primal variables (i.e. the variables of the original problem) as well as for the dual variables (i.e. the Lagrange multipliers for the bound constraints).

Adopting the same approach, Itle et al. handle the inequality constraints that need to be met in the optimisation of a Chemical Vapour Deposition (CVD) reactor modelled using the CFD package MPSalsa (Itle et al., 2004). The optimisation framework applied in this case is the tailored nonlinear programming strategy (as opposed to the adjoint strategy which requires access to the Jacobian and its transpose) based on a reduced Hessian SQP algorithm. The objective is the optimisation of operating conditions of a CVD reactor for the production of a thin film of GaN of spatially uniform thickness. The inequality constraints refer to the maximum thickness of the GaN film. They are handled using a constraint aggregation function (KS function), which replaces all inequality constraints by a single one which can further be incorporated in the objective function as a penalty, thus eliminating all constraints.

Leineweber et al. (2003) have presented an efficient rSQP method for large-scale dynamic optimisation. It follows the tailored paradigm and combines multiple shooting with reduced Hessian SQP. The authors concern themselves with the multistage optimal control problem based on differential algebraic equations. In such problems, the horizon is discretised into a specific number of intervals, which define model stages. Furthermore, the problem can be discretised with multiple shooting, leading to a structured (typically large-scale) nonlinear optimisation problem, which has continuity and path constraints, as well as stage transition conditions and linearly coupled multipoint constraints. Exploiting the structure of the aforementioned NLP, decomposition can be performed. The Jacobian of the constraints is a large sparse matrix composed of dense blocks and the Hessian is block diagonal. Following the approach of Leineweber et al., partially reduced SQP is used to project the NLP onto the subspace of the decision variables and thus eliminate all algebraic constraints. The decomposition employed is guaranteed for index one problems (in that the bases are existent) and leads to a significant reduction in case of problems with many
algebraic variables and is based entirely on local information. The reduced Hessian is updated in blocks using a heuristic condition based on the BFGS formula. The authors claim that the proposed method leads to NLPs of relatively low order, although the dynamic model is explicitly discretised – thus coupling advantages from direct and sequential techniques. Moreover, the efficient use of existing simulators is enabled and the sparsity of the underlying problem is exploited.

Methods for large scale nonlinear programming also include the works of Luna-Ortiz and Theodoropoulos (Luna-Ortiz and Theodoropoulos, 2005; Theodoropoulos and Luna-Ortiz, 2006), which could be considered as part of the equation-free framework and will be properly referenced in a later section. For reasons of completeness though, let it be mentioned that those methods combine the Recursive Projection Method with the reduced Hessian method, resulting on a two-step projection scheme, which has been applied both for steady state and dynamic optimisation problems which are based on process models in the form of input/output simulators.

Armaou and Christofides (2002) present a suite of dynamic optimisation methods of dissipative DPS. The method of weighted residuals with global basis functions is employed for spatial discretisation and are used in conjunction with approximate inertial manifolds. These functions are the (analytical or empirical) eigenfunctions derived using Proper Orthogonal Decomposition. If the number of basis functions that need to be used to capture the dominant dynamic is small, the basis functions are usually used as weighting ones. The resulting low-order dynamic optimisation problem is constrained by ODEs and algebraic relations. Further, temporal discretisation can be performed using appropriate techniques such as backward/forward Euler or central differences, leading to an NLP that can be solved via a conventional reduced Hessian optimiser such as MINOS, or even better a global optimiser. Fast modes are typically neglected; however they may be included in the formulation as long as they are significant (until they become enslaved to the slow ones).

A popular deterministic method for large-scale nonlinear mathematical programming is the Conjugate Gradients (CG) method (Nocedal and Wright, 1999). The linear CG is a well known, well formulated and widely used iterative method for the solution of linear systems of equations, being particularly efficient for large systems. It can be extended to a nonlinear CG method, suitable for large-scale NLP (Fletcher and Reeves, 1964). Fletcher and
Reeves introduce a linesearch to the linear CG algorithm and replace the objective of the solution of the linear system of equation with the gradient of the objective function of the NLP, in order to enable CG to perform nonlinear optimisation tasks. The resulting algorithm is quadratically convergent. Based on nonlinear CG, many variants have been presented and it has been applied to various classes of problems.

Alexeev et al. (2009) explore various large-scale optimisation methods and compare their efficiency for inverse Computational Fluid Dynamics problems. The methods they consider is nonlinear Conjugate Gradients, a quasi-Newton minimisation algorithm incorporating Broyden-Fletcher-Goldfarb-Shanno (BFGS) update procedures as well as a limited memory version of the latter (L-BFGS), a Truncated Newton method and a hybrid algorithm combing L-BFGS and T-N. The case study the authors work on is the parameter identification of a system governed by Navier – Stokes equations. The problem is ill-posed. Specifically, the objective is the estimation of inlet parameters from a set of outlet data. The adjoint optimisation problem is formed and solved using the aforementioned methods, as its solution is faster for a large number of decision variables. The authors show that L-BFGS, T-N and the hybrid strategies exhibit superior behaviour in comparison to that of the CG. BFGS provides an alternative for narrow ranges of decision variables.

A Large-Scale System Hierarchical Optimisation Neural Network (LHONN) has been introduced, which is comprised of a co-ordination network (CNN) and Local Optimisation Networks (LNN), working simultaneously (Hou, 2001). The first is an upper level network, while the latter are lower layer subsystems. The strategy employed for the solution of the large-scale optimisation problem using the neural network is decomposition and co-ordination. The separable augmented Lagrangian is separated into N sub-Lagranian systems, where N is the number of variables. The proposed algorithm does not destroy the separable structure for hierarchical optimisation, i.e. the separability of the Lagrangian function. The neural network is comprised of a coordination network and local optimisation networks and in this approach coordination and local optimisation tasks are performed simultaneously, thus eliminating the waiting time.

Stochastic optimisation methods have also been applied for large-scale systems. Methodologies include genetic algorithms (Rivera, 2001), simulated annealing (Brusco, 2006), differential evolution (Das and Suganthan, in press 2010), radial basis functions (Regis, 2011) and particle swarm optimisation (Kameyama, 2009). The (review) articles
mentioned and the references within them provide various stochastic methods suitable for large-scale systems.

Stochastic optimisation methods are simple to implement and are not trapped in local optima as often as deterministic optimisation algorithms. Most of them are also numerically robust, making few or no assumptions regarding the underlying system (e.g. smoothness). They are also appropriate for black-box simulators. However they do not provide guarantees on attaining convergence to an optimum. Therefore, the systems they are usually applied to are mainly multimodal, non-smooth, they include integer variables and so forth. For example, Exler et al. (2008) apply a tabu search metaheuristic algorithm for the plant-wide industrial process control of a wastewater treatment plant, and the Tennessee Eastman process. The optimisation objective and the mathematical programming problem contains both continuous and integer variables. An SQP Trust-Region strategy is incorporated in the optimisation method applied. The Tabu search component of the scheme helps avoid entrapment to local optima. Multi-objective optimisation has also been tackled with stochastic optimisation methods. To this end for example Sendin et al. (2006) apply genetic algorithms. The thermal sterilization of canned foods is the system which is examined in the aforementioned paper. Two processes for sterilization are considered, with constant and variable retort temperature. The objective is the simultaneous minimisation of the overall process time and maximisation of the retention of certain quantities. The normal boundary intersection and the weighted Tchebycheff method have been combined to a hybrid strategy which provides Paretto-optimal solutions.

So far, various optimisation methodologies have been mentioned, which are appropriate for large-scale NLP. Nevertheless, historically the first approach towards large-scale optimisation was to use surrogate models (also termed metamodels), in order to replace the full complex model of the underlying system with a simpler one (Biegler et al., 2000a). This methodology is used even today (e.g. in Gadalla et al., 2003; Henao and Maravelias, 2011). One can either use a shortcut model and validate it with a detailed one, or alternatively use experimental or simulation data in order to obtain an unstructured model which has the same behaviour as the original one. Neural networks are popular choices for surrogate models.
Irrespective if the model used for optimisation is detailed or simplified, the simplification and/or decomposition of the optimisation problem is always advantageous. For example, analytical thinking may lead to the reduction of the constraints, which not only reduces the dimensionality of the system and consequently the computational cost of mathematical programming, but also potentially reduces its stiffness. An example of simplifying the optimisation problem has already been mentioned: in dynamic optimisation with the RPRH method. Multiple shooting is employed for temporal discretisation. The basis of the low-dimensional subspace of decision variables is a block matrix, comprising of bases for the time intervals (Theodoropoulos and Luna-Ortiz, 2006).

A popular strategy appropriate for dynamic optimisation problems is Control Vector Parameterization (CVP) (Vassiliadis et al., 1999). This strategy results in reformulating the original problem to a finite-dimensional NLP. This is achieved by discretising the decision variables: their value is considered constant or linear within each of the time intervals of the discretisation. The resulting NLP incorporates an initial value problem in which the values for the decision variables are set. The optimisation problem may therefore be seen as a superstructure to the integration scheme. Vassiliadis et al. (1999) propose solving the NLP exploiting exact gradient and Hessian information computed from the sensitivity equations of the integrator. The CVP technique has been applied for many systems of interest to the Process Systems community and comprises an approach which is suitable both for large and small systems.

Another approach to optimisation of complex, interacting systems is to decompose the mathematical programming problem into smaller ones. Optimisation problems for each of the subsystems are formulated and the optimum for the full system is computed sequentially. In each iteration, each of the subsystems is optimised independently and a set of optimal values of decision variables for the overall system (e.g. a site) is estimated by combining the values found. The full system is simulated and the objective function value is updated. (Zhang and Zhu, 2000).
2.2 Libraries for Nonlinear Programming

Scientific and engineering practice (as well as common reasoning) dictates the use of as many of the ready-made tools as possible. This simplifies the assignment at hand and allows the direction of efforts to tasks other than development, which could potentially be more challenging and/or more involving. Besides, typically, ready-made tools have been proved to be accurate and efficient, whereas an in-house developed code would require verification, calibration and optimisation after development. This practice is followed irrespective if the assignment is on simulation, analysis, optimisation or control.

Perhaps the most widely used large-scale NLP code is SNOPT (Gill et al., 2002). It is implemented in FORTRAN and can be called from FORTRAN, C/C++ or MATLAB® codes. It is also part of many powerful optimisation packages, such as AIMMS (which parenthetically is free for academic use), AMPL (Gay et al., 2002), GAMS (Gill et al., 2000), TOMLAB (Holmström et al., 2008) and others. Implementations of SNOPT are also included in some engineering software packages, such as COMSOL Multiphysics (2005). It is also available as part of the NEOS platform (Baldea and Daoutidis, 2006), the latter being a free online server, to which optimisation problems formulated in GAMS can be submitted and is offered by the Argonne National Laboratory of the United States of America. SNOPT is particularly efficient for large-scale equality- and possibly inequality- constrained optimisation, which involves a high computational cost for function evaluations. It employs a reduced space, active set SQP method and uses augmented Lagrangian merit functions for the handling of the constraints. Smoothness of the functions is a requirement, but explicit provision of gradients is not, as those can be computed with finite differences by the optimiser.

Another popular optimiser for large-scale NLP problems is IPOPT (Wachter and Biegler, 2006). It is a primal-dual interior-point algorithm and employs filter line-search. The algorithm is globally convergent to a local optimum and its convergence properties have been thoroughly characterised. It employs second order corrections in order to reduce the infeasibility for a given search direction by performing a convergence step for the constraints. It also uses inertia correction, which requires from the iteration matrix to have as many positive eigenvalues as the number of variables, as many negative eigenvalues as the number of equality constraints and no zero eigenvalues. It employs heuristics to accelerate convergence and it has been shown to perform well in comparison to KNITRO
and LOQO, which both employ interior point schemes both in terms of iterations and function evaluations.

KNITRO is an optimisation package, written in C language code, which is suitable for large-scale optimisation (Waltz and Nocedal, 2002; Byrd et al., 2006). It is an integrated package, including both interior point and active set strategies, which are combined in a “crossover algorithm”. The latter is implemented in two versions, one being direct and the other indirect, based on the Conjugate Gradient method, whereas the active set methodology is based on Sequential Linear Quadratic Programming and an indirect CG solver. Inequalities are handled following the EQP paradigm.

CONOPT is a package for smooth, large-scale, nonlinear optimisation problems (Drud, 1985; Drud, 1994). It is based on a Generalized Reduced Gradient interior point method. It employs linear algebra methodologies to exploit the sparsity of the underlying problem and subspace minimisation methods. It is able to perform both steady-state and dynamic optimisation tasks.

Another popular large-scale LP and NLP optimiser is MINOS (Murtagh and Saunders, 1998). It focuses mainly on sparse problems for which gradient evaluation is relatively inexpensive. For LP problems, a primal simplex algorithm is used. In the case of a nonlinear objective function, but linear constraints, a quasi-Newton approximated reduced Hessian is used within a reduced gradient method. In the presence of nonlinear constraints, a projected Lagrangian method (sparse SLC) is used.

LOQO is an optimisation package employing an infeasible, interior-point SQP method with line search, appropriate for large-scale problems (Benson et al., 2004). The algorithm introduces slack variables to handle inequality constraints. A quasi-Newton’s method is used to solve the first-order optimality conditions, which results in the construction of an iterative computation of estimates for the optimal values of the variables.

LANCELOT is a FORTRAN package for large-scale optimisation (Coleman and Conn, 1984). It employs an augmented Lagrangian method. For problems with simple bounds, a trust region method is employed. Problems with general constraints are tackled using penalty merit functions to reduce them to problems with simple bounds and then use the trust region method. Both direct and indirect (CG) solvers are included.
Veltisto is an optimisation library for large, smooth, constrained optimisation problems, currently under development (Biros, 2010). It implements the Lagrange-Newton-Krylov-Schur (LNKS) method (Biros and Ghattas, 2006a). It is an extension of the “Portable, Extensible Toolkit for Scientific computation” (PETSc). PETSc can be considered as a test-bed, on which parallel, non-trivial PDE solvers can be developed (Balay et al., 2003). It aims at reducing the solver developing time and required effort, rather than be used as a PDE solver package. The debugging and code optimisation tools provided (memory management, profiling, error-tracking utilities) are utilized in veltisto. The latter is mainly targeted to PDE-constrained problems and it adopts a primal-dual interior-point LNKS approach for bounds on the independent variables. It is based on a symmetric variant for quasi-minimum residual Krylov solver (QMR). The merit functions that can be used in veltisto are of $\ell_1$ penalty and augmented Lagrangian type.

### 2.3 Inequality-constrained, large-scale optimisation

The role of constraints to an optimisation problem is to render the identified optimum, as a feasible solution to the underlying system. That means that constraints bind the optimisation procedure with the physics of the system at hand (expressed via a model) and with any engineering limitations (limiting conditions in construction or operation). Since equality constraints ensure the feasibility of the solution by satisfying the model, they should be included in all design optimisation problems. Nevertheless, the second source of constraints to mathematical programming problems is not always available. In many problems, the limiting conditions expressed via inequalities are reduced to simple bounds for variables. Those cases do not pose a problem as optimisers, which are suitable only for equality-constrained problems are able to handle bounds. Some classes of inequality constraints (such as ratios of variables) can be seamlessly reduced to simple bounds on existing variables (Farias et al., 2007) or alternatively in some classes of problems (such as the parameter estimation problems with the errors in variables method) the inequality constraints can be directly incorporated in the model parameters (Tjoa and Biegler, 1991). In this section, the handling of nonlinear inequality constraints in the context of SQP and rSQP will be discussed. It is obvious that those approaches can be applied for linearly
inequality constrained systems; nevertheless there has been significant research in the field of NLP with only linear inequalities (Forsgren and Murray, 1997; Zhu, 2005a; Zhu, 2005b).

There are two main approaches in handling inequality constraints within the SQP framework: the first approach involves the sequential solution of inequality-constrained QP subproblems (IQP) and the second equality constrained ones (EQP) (Nocedal and Wright, 1999). Following the IQP rationale, at every iteration of the SQP (termed outer or major iteration), the nonlinear inequalities are linearised and considered in the QP subproblem, which in turn is solved using an active set approach. This approach tends to lead to quite computationally expensive QPs, the cost of which can be reduced by warm-starting, i.e. initialising the quadratic optimiser with information from the previous iteration. Conversely, in the EQP formulation, at every major iteration an estimation of the active subset of the inequalities is identified using estimates of the Lagrange multipliers and passed on to the QP as a working-set. This method has the advantage of lower computational cost and of the utilization of simpler algorithms for quadratic programming. Both approaches have advantages and disadvantages, however none of the two is appropriate for large-scale NLP (Wright, 1990).

Biggs (2008) discusses how to extend equality-constrained algorithms for inequality constrained problems. The author first outlines the differences between equality constrained QP problems and inequality constrained ones, focusing on QPs involving positive definite Hessians, reduced gradient methods for simple bounds and utilization of penalty functions, providing algorithms and examples. Furthermore, building on the provided information he considers SQP algorithms with inequality constraints, outlining differences between approaches.

Schulz and Bock (1997) have proposed a partial reduction concept to Sequential Quadratic Programming, which is also appropriate for inequality-constrained problems. A more comprehensive presentation is warranted by Schulz (1998). PRSQP can be seen both as a subset and a superset of SQP methods, i.e. SQP can be seen as generalization of PRSQP or the other way round. The method focuses on systems including equations, which cannot be considered for the identification of the null subspace, or inequality constraints. In both those methods, the use of conventional reduced Hessian methods is not possible. PRSQP considers only reducing the QP subproblem to the null space of whichever equality constraints can be used for this matter, passing on linearised equality and inequality
constraints to the QP. Thus, it can be argued that it extends the range of applications of rSQP methods significantly.

In addition, Liu (2009) has proposed a reduced SQP algorithm for inequality constrained NLPs. The author does not assume or force the reduced Hessian to be positive definite, an assumption often made to ensure feasibility of the underlying QP problem. Instead, the search direction is calculated using an augmented inequality-constrained QP, which includes a penalty parameter and slack variables. Depending on the size of the active set, a projected reduced Hessian according to a BFGS scheme can be used, or alternatively if that is not positive definite, another matrix is used, computed in a way that forces it to be positive definite. The algorithm is shown to be globally convergent by employing line searches and converges R-linearly or even Q-quadratically if the required conditions are relaxed.

It has been argued that active set methods exhibit difficulties when applied to large-scale NLPs with inequality constraints (Albuquerque et al., 1997). Albuquerque et al. use interior point methods to exploit the structure of the underlying problem. The authors implement a predictor-corrector scheme, which incorporates an affine scaling predictor step for the search directions and subsequently a corrector step including a second order correction term involving a cantering term, which is computed adaptively. Line search is employed to compute the step length. They apply their approach to control problems, both optimal and model predictive control.

Later, Ternet and Biegler have followed a similar approach for the solution of large-scale NLP problems including NLPs (Ternet and Biegler, 1999). A primal-dual interior-point method is presented to reduce the computational cost, incorporating higher order correction terms to enhance convergence. The proposed algorithm consists of a predictor step, a linesearch and a corrector step. The associated linear systems are decomposed following the reduced space rationale. Full and augmented methods are considered and comparisons between strategies are performed for NLPs of various sizes. Various higher-order methods are considered in this context and compared.

Orozco and Ghattas have proposed a reduced SAND method for NLP based on the reduced Hessian method (Orozco and Ghattas, 1997). One Newton step is used on the state equations to find the search direction in the state variables, whereas a reduced QP problem is solved to obtain the search direction in the space of the decision variables. A line search
is performed before updating the decision variables and subsequently the dependent ones. These updates involve both the range and the null space components of the search direction. The algorithm has been shown to be computationally more efficient than a NAND implementation, even if warm-starting is employed; for the structural optimisation case studies presented by the authors, the decrease in computational cost varies from 50% to 79%, whereas the memory requirements are the same.

Finally, Bartlett and Biegler have proposed an active set QP strategy which enables rSQP to handle inequality-constrained systems (Bartlett and Biegler, 2006). The authors term this methodology QPSchur, as it utilizes Schur complement for dual-feasible QP with positive definite Hessians. They utilize this QP optimiser in the context of rSQP, which is shown with computational experiments to perform better that other QP solvers in the same context.

Mathematical programming problems which incorporate inequality constraints include among others shape optimisation problems (Diehl et al., 2002), as well as process design problems, for example optimisation of chemical vapour deposition processes (Itle et al., 2004) and thin film growth processes in general (Christofides et al., 2009). Another class of problems which inequality constraints naturally arise is dynamic optimisation and optimisation-based control (e.g. NMPC) (Schäfer et al., 2007). Applications include particulate systems (Shi et al., 2006), tubular reactors (Jäger et al., 2007), as well as polymerization reactors (Zavala and Biegler, 2009a) thin film growth (Varshney and Armaou, 2005), catalytic distillation columns (Jäger et al., 2007) and so forth.

### 2.4 Control of distributed parameter and multi-scale systems

Various control methodologies have been proposed over the years. Control algorithms can be classified as classical or advanced (Gopal, 2002). The former refers to of the earliest approaches, and even now perhaps the most commonly used technique: (Proportional-Integral-Derivative (PID) control. It employs linear single-input single-output (SISO) models and its design is based on a (Laplace) transformation to the frequency domain. Even today, classical control has zealous supporters, both in industry and academia (Skogestad, 1997). Modern control strategies include predictive, nonlinear, adaptive and robust control
(Shinners, 1998; Roffel and Betlem, 2004; Kwon and Han, 2005). Monitoring and assessing the performance of a control system is not a trivial task and is one which affects the choice of control methodology (Yuan et al., 2009). Among the advanced control methods the most widely used in industry is Model Predictive Control (MPC) (Qin and Badgwell, 1997; Henson, 1998; Mayne et al., 2000). MPC should be considered more of a suit of methods rather than a single strategy. What those techniques have in common is that the control action is computed from the solution of an optimisation problem. MPC employs a multivariate state-space model of the underlying system, which could be linear or nonlinear, in discrete or continuous time. Typically the objective of MPC, as of any controller is the minimisation of the output from a given setpoint over a finite predictive horizon. The ability to naturally handle constraints within this framework is one of the main advantages of this methodology (Camacho and Bordons, 2004).

The control of DPS is a challenging task. Due to the underlying complexity of the system the use of algorithms designed for lumped parameter systems is often prohibitive. This could potentially be due to the significant spatial variations of the control variables and/or the need/ability to implement control actions in a distributed fashion using appropriate actuators (Armaou and Christofides, 2002).

Control of DPS may focus on controlling spatial profiles (for example in crystal growth processes, etching or chemical vapour deposition reactors), size distributions (for example in crystallization, cell structures and aerosol production), fluid flows (for example in mixing and drag reduction) and material microstructure (for example in thin film growth and nano-scale coatings processes) (Christofides, 2001). Recently, a review of the control approaches in control of DPS has appeared (Padhi and Ali, 2009). Although it is not complete, it gives an account of important developments in the field at hand, starting from the 1960s and reaching the late 2000s. The authors tend to focus more on conference presentations.

The control of DPS has been approached from various routes. The most straightforward of them is to discretise the infinite-dimensional governing set of equations on a mesh using an appropriate method (Finite Elements, Finite Differences, Spectral Elements or Orthogonal Collocation) and to utilise the discrete, finite-dimensional model in the context of a control strategy. Given that the underlying model will most likely be nonlinear, a popular control strategy is nonlinear MPC (NMPC). An example of such an implementation is given by Patwardhan et al. (1992). The authors apply NMPC to two DPS systems: a packed-bed
distillation column and a fixed-bed catalytic reactor. Several assumptions are made to reduce the complexity of the underlying systems. The use of NMPC is advantageous both intuitively and practically, as it can explicitly tackle issues such as change of sign of the gain, inverse response, significantly varying time-delays and maximum satisfaction of the control objectives. Both of their example systems exhibit those characteristics; therefore the superior behaviour of the nonlinear controller in comparison to linear ones, which is shown in the article at hand, is not surprising.

The use of nonlinear control techniques is usually avoided in industry. Linear controllers are simpler and can be implemented on low-cost equipment. One of the most straightforward approaches to enable linear control for nonlinear systems is to perform Jacobian (Lyapunov) linearisation around a desired stationary state. This is equivalent to performing a Taylor expansion around a setpoint, ignoring second- and higher order terms. This approach has been compared to other strategies for deriving control Lyapunov functions (CLFs) namely global linearisation, frozen Riccati equation and quasi-linear parameter varying methods. It has been used in the context of control with Sontag’s formula or using MPC by Yu et al. (2006). The authors show that for the control of a ducted fan the performance of quasi-linear parameter varying methods is superior, with the global linearisation to yield the worst results.

The previous work came to the conclusion that linear control resulted in unstable closed-loop dynamics; therefore a nonlinear technique should be employed. This is a commonly noted scenario in systems which are either highly nonlinear, or are operated over a wide range of conditions (e.g. batch systems). A remedy, which enables the use of linear control, even in such systems, is multiple model predictive control. Özkan et al. (2003) demonstrate this by applying it for the control of a methyl methacrylate vinyl acetate solution copolymerization industrial CSTR. They employ a different model for MPC, depending on the region of operation. The various models they use are obtained by linearisation on the trajectory of a nominal system. The linearisation points are determined by minimising the norm of the difference of the state from a predicted state using the sequence of models. In this case, using one to four models is considered. It is shown that the more the models are used, the less the off-specification product is produced. The largest difference is observed between using one and two models, where the amount (weight) of off-specification product is reduced by more than 55%.
The approach of Özkan et al. (2003) follows the gain-scheduling concept, which determines the bank of models and their space of validity offline and does not offer flexibility in determining which model is more appropriate for the current state in real-time. It is obvious that none of the models perfectly describes the process. Changing from one model to another may cause problems, due to the lack of smoothness in the effective model. Proper initialisation is required to avoid discontinuity in the manipulated variables profile. Recently, Kuure-Kinsey and Bequette (2009) used augmented state information, in conjunction with disturbance estimation and step input disturbance estimation in each of the models in the model bank in order to tackle the aforementioned problems. This leads to a predictor/corrector scheme, in which the model is augmented by disturbances expressing uncertainties.

An alternative approach towards avoiding solving a nonlinear programming problem is to perform linearisation around a reference trajectory and to use the resulting linear model in the context of a linear control strategy, for example linear MPC (Li and Biegler, 1989). This strategy is relevant to the one presented in Chapter 5. Linearisation is performed online, based on a reference point, which is found from the evolution of the system of the previous sampling time, with unaltered control variables. The algorithm that Li and Biegler present is multistep, so the matrices of the state-space formulation change within the horizon considered and the resulting (nonlinear) model is intuitively close to the impulse and to the step response models. The underlying optimisation problem is quadratic and therefore exhibits all the advantages of linear control systems mentioned above. This approach can be used for receding horizon or optimal control, of systems with or without time-delays.

Although it has been reported that it is advantageous to use first principles models in the context of control (Henson, 1998), perhaps the most widely used method is the exploitation of system identification techniques to obtain a linear or nonlinear model which can be used in the context of control (Zhu, 2001). An example depicting the role in system identification for control, constitutes the work of Panos et al. (2010). The objective of the authors is to devise a dynamic optimisation and control procedure for hydrogen storage tanks. The system is comprised of a metal-hydride reactive bed, coupled with three heat exchangers. The authors use model identification techniques to obtain a linear model with four states, one input and one output, which they subsequently use for Model Predictive Control. In order to further reduce the online cost of the control algorithm, multi-
parametric MPC is used. In other words, the optimisation problem is solved offline and the control law is calculated explicitly as a function of the parameters of the system. This procedure results in a number of critical regions, each one of which defines a control law and can rely on a single LTI model, or a piecewise affine (PWA) model (Kvasnica et al., 2004). mpMPC trades online computational cost with memory requirements, as the expression for the control laws need be calculated offline and saved in a database (Pistikopoulos, 2009). This approach can reduce the online computational cost for the control of DPS, thus hurdling a significant obstacle. Let it be noted though, that the number of critical regions can potentially be large, even for small (identified) systems; for original nonlinear systems that reduce to a large LTI model, this approach becomes prohibitive.

Another example of using system identification techniques for control of DPS is given in a recent paper by Harnischmacher and Marquardt (2007). The authors use block-structured models in order to replace the original model of the process at hand. One of the advantages of this approach is that the online solution of an underlying NLP is avoided. The computational cost is reduced by exploiting the structure of the identified model for the computation of sensitivities. The authors also claim that the use of such models enhances flexibility. They consider Weiner, Hammerstein and Uryson models (Xuefeng and Seborg, 1994; Norquay et al., 1999) both for SISO and MIMO systems and they apply the proposed method for the control of an FCC unit, also comparing it with rival strategies, namely gain-scheduled MPC and nonlinearity inversion controllers, both of which transform the nonlinear controller to a linear one.

A different trend is to replace the nonlinear models with stochastic counterparts or hybrid models. A popular approach is to use neural networks. It has the advantage that it is purely data-driven and thus does not presuppose a model for the underlying process. Such an approach has been presented by Aggelogiannaki and Sarimveis (2008). They outline two formulations, both data-driven, the first using an artificial neural network to construct a model for the process and the second relying on the intrinsic separation of time and space dynamics. In specific, any feasible solution can be considered as a product of a finite number of basis functions in the spatial domain and some time coefficients; they use Singular Value Decomposition on the data in order to obtain spatial basis functions which span the solution space, whereas the time coefficients are given from a neural network, which was trained using the data provided. This approach bears similarity to and is along the lines of the Principal Component Analysis.
Passivity and thermodynamics have also been used for the control of DPS. The notions of entropy and dissipation are related to availability and entropy production respectively (Alonso and Ydstie, 2001). Entropy is defined as a concave, first-degree positively homogeneous function. Intensive variables are tangent to entropy. A non-negative storage function of entropy and the intensive variables can be defined using a pair of points. This function takes the value zero if the points are in equilibrium. It is a measure of a distance from a fixed reference point (Balaji et al., 2010). Intensive variables provide feedback signals, closing the loop for control.

In this Section, the focus has been on different control strategies, which exploit the features of, and are appropriate for, distributed parameter systems. Research has been also directed towards designing new optimisation algorithms, appropriate for this class of problems. Most of the large-scale NLP methodologies can be (and most of them already have been) applied for optimisation-based control. Using large-scale NLP optimisers enables the use of rigorous models for control (Zavala et al., 2008).

One very interesting work following this paradigm was recently presented by Zavala and Biegler (2009b). The authors concern themselves with the control of a multi-zone low-density polyethylene (LDPE) tubular reactor and they conclude that the model state can be reconstructed from a limited number of temperature gauges. Among some of the issues arising in such reactors is the continuous (albeit largely unpredicted) polymer deposition on the reactor walls and the poor heat transfer properties of the fouling layer. They employ NMPC using a fully discretised model of the reactor which produces a large scale NLP problem. This is solved using the interior-point IPOPT optimiser, which was outlined in Section 2.3. The observability of the model is quantified by monitoring the projection of the reduced Hessian evaluated at the point which was identified as optimum.

2.5 Model reduction methods for control systems

It has already been established that control of DPS constitutes a non-trivial task. Nevertheless, it has been observed by many researchers, that although the formulation of such problems could be described as intimidating due to the large number of variables
involved, their dynamics are usually characterised by a finite number of significant components. This observation has been exploited in many ways, both following systematic and heuristic paradigms. Systematic attempts for the control of DPS include the utilization of model reduction technology in order to retrieve a low-dimensional model, which can further be used for linear or nonlinear automatic control. Model reduction methodologies which have successfully been applied in conjunction with control techniques include Galerkin’s method (Shi et al., 2006), Proper Orthogonal Decomposition (or Karhunen – Loeve expansion) (Shvartsman and Kevrekidis, 1998; Shvartsman et al., 2000; Armaou and Christofides, 2002) approximate inertial manifold methods (Christofides and Daoutidis, 1997; Ito and Kunisch, 2008), reduced Lagrange basis methods (Ito and Ravindran, 1998), modal decomposition (Dubljevic and Christofides, 2006), and the equation-free methods (Armaou et al., 2004b; Armaou et al., 2005; Siettos et al., 2006).

Model reduction has been employed in the control systems design since the 1970s (Wilson, 1970). The objectives vary: one may want to obtain a reduced model so that the resulting controller would be low order as well, or may employ model reduction techniques to isolate the desired part of the system dynamics. Difficulties in controller design arise since the controller is part of the closed loop formulation providing the control actions that need to be as less perturbing (i.e. of low energy) as possible. The objectives of controller reduction do not necessarily coincide with those of model reduction (Obinata and Anderson, 2000). This section will provide an overview of model reduction methods for control purposes, starting from methodologies for linear systems and then continuing to nonlinear systems. The former will mainly be reviewed for completeness, as they are tangential to the scope of this work. Nevertheless, they have been applied to several classes of nonlinear distributed parameter systems (Moheimani et al., 1997; King and Hovakimyan, 2003).

Truncation-based model reduction truncates state variables by partitioning the system matrices. The transfer function of the full and reduced systems have the same outputs for infinite angular frequency (\(\omega=\infty\)), whereas there exists a discrepancy for zero frequency (\(\omega=0\)). A form of truncation is mode truncation, which forces the reduced system to retain some of the eigenvalues of the full system. One method for accomplishing that, is by utilizing aggregation techniques (Bultheel and Barel, 1986).
Balanced truncation chooses which states realizations to truncate by demanding that the reduced model would have equal controllability and observability grammians, in other words the contribution of controllable and observable states to the response of the reduced system would be the same. For a stable system, the resulting reduced one would be asymptotically stable, controllable and observable. The discrepancy of the full and reduced system (expressed via the infinity norm of the error in the frequency domain) is bounded. Let it be noted that a large $H_2$ error norm does not imply that the $H_{\infty}$ norm is of the same proportion. (Safonov and Chiang, 1989; Obinata and Anderson, 2000)

Singular perturbation is an alternative method of model reduction for controller design. It aims at truncating the fast variables. Contrary to the balanced truncation, the full and reduced models match for zero frequency but do not for infinite. It can be argued that truncation methods are a subset of generalized singular perturbation (Obinata and Anderson, 2000; Walsh et al., 2001)

Other methods result in the computation of a reduced system by minimising an approximate error norm, such as the $H_2$ and the $H_{\infty}$. This procedure is performed numerically rather than analytically. Nevertheless, if the $L_2$ error norm is considered, the reduced model can be computed from linear algebraic equations. Another model reduction method is based on the Hankel norm approximation. This norm is a function of the controllability and observability grammians and is the $L_2$ norm of the Hankel operator, the latter being used to express future outputs of a linear time-invariant system as a function of past inputs using impulse responses. Optimal or suboptimal implementations of Hankel model reduction can be formulated, the former resulting in the lower error bound (Obinata and Anderson, 2000; Schilders et al., 2008).

Several model reduction methods exist which do not start with the assumption that the model is linear and have been applied for controller design. Recently, some very interesting review papers for those methods have appeared (Christofides, 2001; Kadam and Marquardt, 2007; Li and Qi, 2010; Theodoropoulos, 2011). In the first and the last review, Christofides and Theodoropoulos respectively recapitulate the research efforts of theirs and their collaborators. On the other hand, Li and Qi provide a review of methods, which exploit the spatial-temporal separation for modelling, optimisation and control. They categorize the methods they present depending on their basis: Galerkin, collocation or approximated inertial manifold method. The spatial discretisation can follow the finite
2. Literature review

element, finite difference spectral or Karhunen-Loève paradigm. The authors also concern themselves with a review of methods for the modelling of unknown systems, following equation residual minimisation, prediction error minimisation and kernel-based methods.

Generalized invariants have been presented as a tool for the control of DPS (Norquay et al., 1999). They build on a geometrical viewing of the underlying system of equations and rely on symmetry groups for control. The associated stability analysis is generic and applies to any nonlinear DPS. The main idea is that the invariance of the solution results in stability conditions and eventually the control action. Within this framework, both continuous and discontinuous (switching) control can be considered.

Christofides and collaborators have introduced in the late 1990s a framework for model-reduction based controller synthesis (Christofides, 2001). It has been successfully applied to reactive processes, particulate systems as well as fluid flows. It couples approaches found in systems dynamics and nonlinear control, in order to result in a framework for the design of nonlinear low-order feedback controllers, of fully defined characteristics (performance, stability and robustness). The idea exploited in this approach is the same as in most model reduction methods: there exists a low-order subspace which can be considered as dominant. This subspace corresponds to the slow part of the eigenspectrum. Galerkin’s method is employed to project the model onto this subspace and to derive a low-dimensional set of ODEs which describes the dominant dynamics. This strategy can be combined with singular perturbation, in order to reduce the dimensionality even further by exploiting the separation of the eigenspectrum of the underlying system. The result of this procedure is the construction of an approximate inertial manifold, which is then used for the synthesis of low-order feedback controllers based on geometric control techniques.

Shvartsman and Kevrekidis (1998), in a very insightful paper, deal with model reduction-based control of DPS. They design pole placement and linear quadratic regulating low-order controllers using approximate invariant manifold and Galerkin-Proper Orthogonal Decomposition methods. Although the reduced models are nonlinear, the controllers designed rely on their linearisations around steady-states. In specific, the AIM approach results in the computation of a nonlinear relation for a forced AIM following a nonlinear Galerkin scheme. On the other hand, the POD scheme employed uses a snapshot matrix, i.e. an ensemble of state information obtained using the full model, to extract empirical basis functions and subsequently a nonlinear approximate dynamical system. The more
eigenfunctions are used, the better the dynamics ("energy") of the full system is captured, thus resulting in more accurate reduced model. The authors observe that even a small number of eigenfunctions results in a good approximation of the original system. In this work, the assumption is made that full state information is available. The case study they present is concerned with stabilization of a saddle node, so the point to which convergence is achieved, is predicted rather than set.

In the previous section, multiparametric MPC was noted as an efficient methodology resulting in MP controllers of low online cost. It is based on solving the associated optimisation problem offline. The basis of mpMPC is a linear model of the process and several works have been cited where the model is identified from process data. Here, the problem of designing of mpMPC controllers can be revisited in the light of using systematic model reduction techniques resulting in a low-dimensional linear model which can be exploited in the context of mpMPC. One of the techniques which can be used in this framework is POD (Hovland et al., 2006). The conventional POD methodology may result in a model with undesirable and unpredictable features. A goal-oriented POD method has been proposed by Bui-Thanh et al. (2007) which is more appropriate for the objectives of the specific venture. Hovland et al. have applied both in order to obtain reduced models for use in output feedback control.

Finally, another approach to control DPS based on model-reduction, is following the equation-free concept. Since this approach is particularly relevant conceptually to this work, an exhaustive review of contributions following this paradigm is presented in the next section.

### 2.6 Equation-free methods in optimisation and control

The equation-free methods have been recently proposed in order to enable systems, for which accurate equations in closed form do not exist, to be used in computational system-level analysis. (Theodoropoulos et al., 2000; Kevrekidis et al., 2003a; Kevrekidis et al., 2004; Kevrekidis and Samaey, 2009). The equation-free methods mainly address to micro/nano-scopic and multi-scale systems, which cannot be accurately described by a set of equations
in the system scale (observable scale) in closed form. Closure of a given system is considered available if the model, although only containing system-level variables, can substantiate the effect of the neglected ones. Within this framework, variables are classified as coarse if they can be used within a macroscopic description or fine, if they refer to the smaller lengthscales.

It is outside the scope of this work to give an exhaustive review of the equation-free class of methods; rather, a review of all the classes of problems to which they have been applied will be given and the focus will be on the optimisation and control applications. A very good introduction to the equation-free notion is given by Kevrekidis et al. (2004), where the authors state the objectives of this class of methods, as well as the followed approach. A recent review of the equation-free methods and their applications is provided by Kevrekidis and Samaey (2009). However, before referring to the equation-free methods, some background information need to be given.

The equation-free class of methods was inspired mainly by the Recursive Projection Method (RPM) (Shroff and Keller, 1993). RPM can be used for continuation of steady states, comprising a wrapper for input/output fixed-point simulators. The convergence of those simulators deteriorates or may fail when the eigenvalues of the scheme approach or exceed the unit circle. The iterates of the fixed-point (Picard) iterations can be used for the computation of an approximate basis for the subspace of the (few) eigenvectors belonging to the eigenvalues of largest magnitude, which are the ones inhibiting convergence. Hence, the full problem is decomposed to two sub-problems: One arising from the projection onto this low-dimensional subspace and the other from the projection onto its complement. Newton’s method is used for the former, whereas Picard iterations are being used for the latter. This procedure accelerates convergence and extends the region of convergence to unstable steady-states. A by-product of this procedure is the computation of reduced quantities, such as the Jacobian, which are projections onto the low-dimensional subspace considered. These quantities may, for example, be used for bifurcation analysis (Koronaki et al., 2003) or other system-level tasks. Computing those by-products may be performed without using the equations of the system explicitly, setting the foundation for equation-free methods.

Equation-free can be seen as a suite of computational tools for multiscale systems. It is not restricted to performing coarse time-stepping and coarse bifurcation analysis which have
already been mentioned. One significant computational tool is projective integration, which uses a given input/output simulator to accelerate time integration, by performing extrapolation in time on the coarse variables (Kevrekidis et al., 2003a). An analogous method aiming to extend the spatial domain considered rather than the temporal one, is the gap-tooth scheme (Kevrekidis et al., 2003a; Kevrekidis et al., 2003b). Evolution of the microscopic model is only performed in small spatial domains (called teeth), whereas values for the coarse variables are interpolated over the whole domain. The gap-tooth scheme has been combined with coarse projective integration resulting in a computationally efficient scheme, called patch dynamics, which provides values for system-level variables for all desired spatial and temporal coordinates, although they only use the black/box time at specific points and time intervals (Kevrekidis et al., 2003b). Mention should also be given to micro-Galerkin methods, which aim to enable coarse integration by using a coarse model that is derived from coupling coarse variable values from the timestepper in conjunction with offline identified basis functions for projective integration (Kevrekidis et al., 2003a).

In the rest of this section, equation-free methodologies for steady-state and dynamic optimisation will be mentioned, as well as for control. Steady-state optimisation following the principles of the equation-free framework has been applied by Luna-Ortiz and Theodoropoulos (2005) and Bindal et al. (2006). In both cases, a transient simulator was used as a building block, around which the computational superstructure (wrapper) was developed, employing equation-free model order reduction.

Luna-Ortiz and Theodoropoulos (2005), tackle the steady-state optimisation of large-scale using black-box simulators problem by combing RPM (Shroff and Keller, 1993) with a reduced Hessian method (Biegler et al., 1995). In specific, RPM is exploited for computing a feasible solution of the system and for the derivation of a reduced order representation of the full model. A “by-product” of RPM is a basis for the dominant subspace of the system, i.e. of the invariant subspace belonging to the eigenvalues of the largest magnitude. This basis is exploited to efficiently construct projections of Jacobian and gradient matrices onto the subspace in question in a matrix-free fashion, with a small number of directional perturbations. A reduced Hessian Sequential Quadratic Programming method is employed for optimisation. The main feature of this technique is that the formulated QP subproblem involves a reduced Hessian which belongs to the null space of the equality constraints. The computation of a basis for this subspace, repeated at every iteration, involves inverting the
Jacobian, which is of high dimension for large-scale problems. Luna-Ortiz and Theodoropoulos (2005) consider using the reduced onto the invariant subspace Jacobian for a computation of an approximate basis for the null space of the equality constraints. They illustrate the behaviour of the proposed optimisation scheme using the optimisation of a tubular reactor as a case study.

At a later independent study, Bindal et al. apply the equation-free concept for steady-state optimisation of microscopic systems (Bindal et al., 2006). The authors present two case studies. The first, simpler one is based on an ideal CSTR where a reaction network takes place, which in turn consists of a modified Fuguit and Hawkins mechanism. Results using both continuum and stochastic microscopic simulator are presented. The second case is based on the catalytic oxidation of CO on Pt and is simulated using a lattice/kMC-based transient simulator, where the objective is the maximization of the CO production rate. Both local and global optimisation was considered. The local optimisation techniques applied consist of the Nedler-Mead and Hooke-Jeeves algorithms, which only require objective function values and SQP, which is gradient-based. These are coupled with global optimisation strategies, namely multilevel coordinate search (MCS) and differential evolution (DE), the former being deterministic and the latter stochastic. The authors suggest that selecting system-level variables are crucial for the procedure and observe that increasing the number of particles considered in the microscopic simulator enhances the results, in that they approach to the counterparts using continuum simulators.

The equation-free framework has been also used for the solution of dynamic optimisation control problems. Armaou and Kevrekidis (2005) propose a method of computing the optimal trajectory, in terms of minimal manipulation of the manipulated variables, for a system governed by microscopic evolution laws to fall from a stable steady state to another. The system they examine models the oxidation of CO over Pt with a Gillespie model and is bistable. The equation-free concept enables the use of methods of conventional optimisation techniques (in particular SQP and Hooke-Jeeves algorithm were used). Although these strategies are gradient-based, Jacobians and Hessians are constructed in a matrix-free fashion, exploiting the timestepper for directional perturbations. The final desired state is passed on to the optimisation problem using a penalty. The authors discuss the effect of noise due to the stochastic nature of the simulator as well as the effect of the number of particles simulated.
A similar approach has been presented by Theodoropoulos and Luna-Ortiz (2006). The authors propose a methodology for dynamic optimisation based on the direct multiple shooting method. In this formulation, the infinite-dimensional original NLP is transformed into a large optimisation problem. The time horizon considered is split to intervals, so the dynamic model of the system is transformed to a set of equality constraints for the final mathematical programming problem. Those constraints are imposed to warrant continuity of the continuous trajectory computed and are hence called continuity constraints. The computation of feasible points by applying Newton’s method on those constraints entails the solution of a linearised system, which involves Jacobian matrices with respect to the state and decision variables. In the conventional formulation, the Jacobian is a sparse, block diagonal matrix, possibly of very high dimension. The $j$-th block includes the Jacobian matrix of the system model, which refers to the $j$-th time interval. The authors propose computing a basis for the maximal invariant subspace corresponding to the leading eigenvalues for each of the time intervals. These bases can be assembled into an overall (sparse) basis, which heuristically defines a dominant subspace of the system. The equality constraints can be projected onto this subspace, so that the calculation of a feasible point only involves performing Newton’s method on the low dimensional dominant subspace of the system, whereas Picard iterations are performed to its complement. This concept is combined with the reduced Hessian SQP optimisation strategy to yield an efficient dynamic optimiser.

The equation-free framework has also been applied for feedback control tasks. A first, brief introduction of the equation-free concept to control problems has been given by Siettos et al. (2003a). They design offline a linear feedback controller, which is based on a coarse discrete time process model only considering the slow variables, as defined by the RPM. They employ a third order time varying FIR filter to smooth the output of the microscopic simulator. They apply the proposed scheme for the oxidation of CO over Pt, simulated using a microscopic kMC-based simulator.

In a later publication, Siettos et al. (2003b) employ state-feedback control on a system for the catalytic oxidation of NO by $H_2$ on Pt and Rh modelled by a microscopic kMC simulator. In each sampling interval, the controller stabilizes the system by computing a gain matrix for which pole placement to desired values is achieved, i.e. the eigenvalues corresponding to the slowest dynamics are directed to specific values. The control problem is a tracking one, the objective of which is to force the system into following an open-loop coarse
solution branch. The manipulated variable is the bifurcation parameter. Pseudo arc-length steps are followed by a local coarse linearisation using the findings from RPM. This analysis presupposes that the linearised, augmented model should be stabilizable, which is the case for systems exhibiting simple limit points.

Siettos et al. (2004; 2006) revisit the NO oxidation case study, modelled using a kinetic Monte Carlo microscopic simulator and design a coarse grained pole placement controller for which they design a nonlinear controller. Feedback linearisation with pole placement in a single step is used, which in general requires access to the system equations; however the authors use it in the equation-free context in conjunction with black-box simulators. In specific, to circumvent the unavailability of analytical equations for the description of the system, power series are used, the coefficients of which are computed around a stationary state using short calls (“bursts”) to the given simulator, either using matrix-free methods or by solving an optimisation problem. The proposed method combines the single step feedback linearisation with pole placement, which avoids the restricting conditions of the conventional two-step procedure, with the equation-free framework.

Armaou et al. (2004a) revisit the 1-dimensional FitzHugh-Nagumo system examined in the introductory paper of equation-free technology by Theodoropoulos et al. (2000), for which they design coarse linear, discrete-time controllers. To test the equation-free formulation, the authors substitute the FHN system with an input/output Lattice Boltzmann simulator employing the Bhatnagar, Gross and Krook approximation (LB-BGK) for the same system and in specific the zeroth moment. The authors formulate third- and fourth-order Linear Quadratic Regulators using a reduced-order linear approximate model which corresponds to the slow dynamics identified using RPM on the open-loop system. These dynamics are identified on the inertial manifold, i.e. when the fast modes have already been enslaved to the slow ones. After convergence to a stationary state is achieved, the actuator effect is evaluated. The resulting linear model can be used for the design of a pole placement controller, or in the context of an optimal control algorithm. The objective of the pole placement control is to stabilize the system without altering the spectral gap which translates to separation of temporal scales.

This approach of designing coarse controllers has been extended to consider gaptooth scheme-based simulators (Armaou et al., 2005). This scheme enables the simulation of larger spatial domains with microscopic or mesoscopic simulators: integration in time is
only performed in specific spatial subdomains (teeth), which are separated by gaps. The smoothness of the solution’s spatial variation is hence exploited in order to evaluate the solution in larger domains. The microscopic variables are not defined in the teeth; the coarse variables in those areas are computed from interpolation using the coarse variable values at the teeth. The black-box simulator is microscopic and is exploited by the wrapper to simulate a distributed parameter system, the wrapper performing the control tasks as well. The controller design follows the algorithm of (Armaou et al., 2004a): an LQR algorithm employing a linear reduced model corresponding to the slow dynamics of the system incorporating the evolution of the slow coarse variables and the actuator effect.

2.7 Summary

Many, if not most, of engineering systems can be classified as DPS. Examples include particulate systems, tubular reactors, polymerization systems, thin film growth, packed bed distillation columns and reactors and so forth. These systems are typically accurately modelled by systems of PDEs. Those sets are of infinite-dimension and are discretised on a computational mesh to produce a finite number of DAEs that can be solved numerically. The resulting set of discretised equations is usually large, so the use of efficient direct or iterative methods for its solution is imperative.

Large-scale nonlinear programming problems arise naturally in the design of DPS. The strategy adopted for such problems is usually a deterministic one, although stochastic methods have also been proposed. Advantages of the latter approaches include the simplicity of the algorithms employed, their intuitive nature and the intrinsic ability to use existing simulators due to their derivative-free operations. However, it has been shown that stochastic methods are less efficient if the search space is large and are therefore more appropriate for problems of modest size. Furthermore, although intuitively they are able to converge to the global optimum, they do not provide any guarantees for convergence.

An approach to avoid formulating large NLPs is to generate simplified models from the full ones, or approximating the latter with unstructured (or block-structured) ones, trading
efficiency of the optimisation procedure with loss of accuracy of the model. To the same end, systematic model-reduction techniques have been proposed, which preserve the most important features of the original model and incur minimal loss of accuracy. A typical attribute exploited for model reduction is the separation of time-scales to slow and fast ones, which is an indication of dissipativity and thus a characteristic of most Process Systems.

Perhaps the most widely used of all deterministic NLP methodologies is Sequential Quadratic Programming. Within the context of SQP, handling of large-scale problems is enabled through low-order projections onto the subspace of the independent variables. This notion gives rise to the so-called reduced Hessian framework, which is more appropriate for problems involving more dependent variables than independent ones. The efficiency of these methods might be compromised due to the need to compute and invert the system Jacobian at every iteration of the optimiser. An accurate approximation of the Jacobian is needed for the computation of a basis for the subspace of decision variables. For large-scale systems the use of state-of-the-art iterative Krylov subspace-based methods would be required, as the inversion a large matrix is an immense task. Hence, the introduction of new algorithms that hurdle this obstacles would be welcome.

Handling of inequalities within the SQP framework can follow the inequality constrained QP (IQP) or the equality-constrained-QP approach, depending on the nature of the constraints of the QP subproblem. Those classes of methods have inspired approaches tailored for reduced Hessian SQP. Both interior-point approaches have been proposed, leading to EQP and IQP-based approaches, relying on partial reduction of the problem, only to the kernel of the equality constraints. The inequalities of the QP subproblem come from linearised nonlinear inequalities of the original NLP. Recently, another approach for inequality-constrained NLP with rSQP has been proposed, which employs slack variables.

Proprietary or open-source, free or commercial software is widely available implementing optimisation algorithms suitable for DPS. However, linking optimisation packages with simulation tools is not always a trivial task. Setting up the interface between the low-level simulation software and the upper-level optimiser could be a tedious task; what is more, retrieving certain information (such as gradient) from the low-level could prove to be a formidable if not impossible task. This limiting factor could potentially inhibit the Simultaneous Analysis and Design procedure and require a Nested Analysis and Design
strategy, where the solution of the optimisation problem involves converging many simulation problems. The integration of engineering packages is an important feature. All in all, the Process Systems Engineering software should provide a dependable decision tool in the service of the Engineer.

Optimisation and control are interlinked, as explained in the introduction and shown from the literature review control is used to drive the process to its optimal operation conditions (tracking control) and to keep it there irrespective of the disturbances (disturbance control). Advanced control algorithms – more often than not – reduce to solving optimisation problems (linear, quadratic or nonlinear). Furthermore, the controllers may be receiving the set-points in real-time from a site-wide real-time optimisation procedure, or even the controllers may have economic constraints (e.g. maximization of a profit) rather than minimisation of the distance from a set-point. Thus, the issues that arise in the control of DPS are similar to, or sometimes same as, the ones one faces in the field of optimisation of DPS. Consequently, it constitutes no coincidence that most NLP algorithms can be – and routinely are – used for optimal or feedback control. Using large-scale NLP optimisers enables the use of rigorous models for control.

MPC seems to be the most popular among advanced control strategies both in industry and academia. The term MPC refers to a suite of methods rather than a single algorithm. This class features a receding horizon, feedback approach to finite horizon optimal control and one of its main advantages is the intrinsic and intuitive handling of constraints. As the control action is computed from an optimisation problem, the computational cost is a major issue in MPC. One approach towards minimising of this cost is to use state of the art algorithms for large-scale NLP. Another one is to use surrogate models, which as in the optimisation case can be limiting. A more systematic viewing of producing reduced models for controls is model reduction.

Model reduction methodologies which have successfully been applied in conjunction with control techniques for DPS include Galerkin’s method, Proper Orthogonal Decomposition, approximate inertial manifold methods, reduced basis methods following the Lagrange approach and modal decomposition. A typical characteristic exploited in many different approaches to model reduction (which consequently result to different reduced order models) is the separation of time-scales. It is shown to be an indication of dissipativity and is exhibited in many engineering systems.
An important class of DPS is multi-scale systems, spanning multiple temporal and spatial scales. Those systems may be modelled using hybrid simulators, resulting from coupling macroscopic (system/coarse-level, deterministic, PDE-based) simulators and microscopic (fine-level, possibly stochastic) simulators. Such structures pose a problem to the conventional methods used for analysis, optimisation and control, as the system equations governing the phenomena at the fine level are unavailable, although intuitively existent. A class of algorithms that leaps over this impediment is “equation-free” framework, which enables performing system-level tasks such as integration, bifurcation analysis, optimisation and control efficiently using the existing simulators in an input/output fashion in conjunction with effective matrix-free techniques.
3. Model reduction-based PDE-constrained optimisation of large-scale dissipative systems

3.1 Introduction

Optimisation is tightly bound conceptually to engineering systems design and operation. Especially with regard to process systems, on which this work focuses, optimisation is involved in process synthesis and design, in operations and in control (Biegler and Grossmann, 2004). In all three categories of applications, nonlinear optimisation problems arise naturally. Probably the most widely used among classes of NLP strategies is Sequential Quadratic Programming (SQP) (Nocedal and Wright, 1999; Edgar et al., 2001). Many variants of SQP exist, building on the main idea that stationary points can be located by applying Newton’s method to the first order Karush – Kuhn – Tucker (KKT) optimality conditions. This leads to a scheme of high rate of convergence that has been shown to require a lower number of function evaluations than rival strategies (Fan et al., 1988).

Most processes can be adequately simulated by models consisting of sets of Partial Differential Equations (PDEs). Thus, PDE-constrained optimisation problems surface in engineering design, as the model for the process provides equality constraints to the mathematical programming problem. In the case of distributed parameter systems, the governing PDEs are typically discretised onto a computational mesh, leading to a large
number of equality constraints and consequently dependent variables. Reduced Hessian SQP methods (rSQP) (Biegler et al., 1995; Cervantes and Biegler, 1998) are more appropriate than SQP for this class of problems. They rely on the elimination of the dependent variables and perform the nonlinear optimisation task required by solving a sequence of reduced QP subproblems. This elimination is performed by projecting the full problem onto the null space of the equality constraints. The resulting scheme is most efficient for problems which involve a small number of dependent variables in comparison to the number of independent ones. Although a significant reduction in dimension is performed, the computational cost may remain comparable to conventional methods (Burger and Mühlhuber, 2003).

One way to reduce the computational cost of the optimisation procedure and therefore render it more appropriate for large sparse systems is to employ Krylov subspace methods (Haber and Ascher, 2001; Biros and Ghattas, 2006a). Another approach is to employ model reduction techniques. Recently, the Recursive Programming Reduced Hessian optimisation framework has been proposed for large-scale NLP utilizing existing transient simulators of dissipative processes (Luna-Ortiz and Theodoropoulos, 2005; Theodoropoulos and Luna-Ortiz, 2006). It extends the concepts of rSQP by exploiting the separation of time-scales, exhibited in many engineering systems, for model reduction.

This chapter presents a novel static optimisation algorithm, inspired by the recent developments outlined above. It combines model reduction techniques with reduced Hessian SQP, combination which results in an efficient scheme appropriate for steady-state optimisation of dissipative large-scale systems. It features a two-step projection scheme, firstly onto – what is perceived to be – the dominant subspace of the system, and secondly onto the subspace of decision variables. An important advantage of this approach is that it enables the use of existing simulators, thus extending their use to optimisation tasks.

The novelty of the work presented here is that it enables the use steady-state solvers for static optimisation which follows the Recursive Projection Method approach for model reduction (Shroff and Keller, 1993). Previous work on steady-state mathematical programming built on dynamic simulators (Luna-Ortiz and Theodoropoulos, 2005), resulting in a significantly different procedure than the one presented here. Note that given a choice, one would more often than not prefer to locate steady-states using a static rather than a transient simulator, since convergence would normally be attained faster.
Furthermore, here some formal mathematical exploration of the effect of model reduction to the optimisation procedure with this class of reduced NLP algorithms is attempted for the first time.

The rest of this chapter is organized as follows: in Section 3.2 an overview of the SQP method will be given and in Section 3.3 the reduced Hessian SQP will be outlined. Those methodologies constitute the basis for the proposed optimisation technique. In Section 3.4 the model reduction technology employed will be delineated, before the proposed method is introduced in Section 3.5. The concepts presented are condensed in an algorithm (3.1). Next, strategies for reducing its computational cost will be outlined (Section 3.6) and used for the extraction of a more numerically efficient algorithm (3.2). In Section 3.7 the results from the implementation of the proposed algorithms to illustrative case studies are presented. Comparison with the reduced Hessian method is made in terms of optimisation paths followed and computational cost. Section 3.8 presents a technique for handling black-box systems, the simulators of which cannot be used directly for eigenanalysis purposes; in these cases the computation of a basis for the dominant subspace is not straightforward. Finally, some concluding remarks are made in Section 3.9.

### 3.2 Sequential Quadratic Programming

SQP has proved to be the most popular among the deterministic local optimisation methods for nonlinear programming (NLP) (Nocedal and Wright, 1999; Edgar et al., 2001). This is partly because it has been shown to converge fast and usually requires the least function evaluations among rival strategies (Fan et al., 1988) and because it is suitable for systems with significant nonlinearities (Nocedal and Wright, 1999). The mathematical programming problem dealt with is the minimisation of a given objective function $f$: $\mathbb{R}^{d+df} \to \mathbb{R}$:

\[
\min_{\mathbf{x}} f(\mathbf{x})
\]  

(3.1)

Subject to the equality constraints:
where \( \mathbf{G} : \mathbb{R}^{Nd \times \text{dof}} \rightarrow \mathbb{R}^{N} \), \( \mathbf{x} \in \mathbb{R}^{Nd \times \text{dof}} \) is the vector of the dependent \((u \in \mathbb{R}^{N})\) and independent variables \((z \in \mathbb{R}^{\text{dof}})\): \( \mathbf{x}^T = [u^T \ z^T]^T \), with \( N, \text{dof} \in \mathbb{N} \). The functions \( f \), and \( \mathbf{G} \) involved in the formulation are assumed to be smooth and at least twice differentiable. Note that Equations (3.1) – (3.2) define a (nonlinearly) equality-constrained NLP problem; inequality constraints and bounds for the variables will be considered later on in the analysis.

SQP solves the underlying NLP iteratively, by solving the KKT optimality conditions using the Newton’s method. To this end, a Lagrangian function, \( \mathcal{L} : \mathbb{R}^{Nd \times \text{dof}} \rightarrow \mathbb{R} \), is introduced, which combines the objective with the constraints:

\[
\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda^T \mathbf{G}(\mathbf{x})
\]

Where \( \lambda \) is the vector of the Lagrange multipliers, \( \lambda \in \mathbb{R}^{N} \).

The first-order KKT conditions for this system (necessary conditions), are:

\[
\nabla \mathcal{L}(\mathbf{x}, \lambda) = \nabla f(\mathbf{x}) - \sum_i \lambda_i \nabla G_i(\mathbf{x}) = 0
\]

\[
\mathbf{G}(\mathbf{x}) = 0
\]

The second-order KKT conditions (sufficient conditions), state that if a set of variables \( \mathbf{x}^* \) and Lagrange multipliers \( \lambda^* \) satisfy (3.4), define a positive definite Hessian \( \nabla^2_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \lambda^*) \) and also \( \mathbf{d}^T \nabla^2_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \lambda^*) \mathbf{d} > 0 \) for all vectors \( \mathbf{d} \) for which \( \mathbf{d}^T \nabla \mathbf{G}(\mathbf{x}^*) = 0 \), then \( \mathbf{x}^* \) is a strict local solution for (3.1) - (3.2).

Applying Newton’s method to the system of equations (3.4), results in the following iteration, from which the variables and the Lagrange multipliers are updated:

\[
\begin{bmatrix} \mathbf{B} & -\mathbf{J}^T \\ \mathbf{J} & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla \mathcal{L} \\ \mathbf{G} \end{bmatrix}
\]

Where \( \mathbf{J} \in \mathbb{R}^{Nd \times N} \) being the Jacobian of \( \mathbf{G}(\mathbf{x}) \) and \( \mathbf{B} \in \mathbb{R}^{(Nd + \text{dof}) \times (Nd \times \text{dof})} \) is the Hessian of the Lagrangian: \( \mathbf{B}(\mathbf{x}, \lambda) = \nabla^2_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda) \). Iteration (3.5) is termed Newton-Lagrange method and is well defined if the matrix at the left hand side is not degenerate. This is true assuming that
the Jacobian has full rank and that the null-space component of $B$ is positive definite (i.e. $d' B d > 0 \ \forall \ d \neq 0 : J d = 0$).

It can be shown that the solution of (3.5) coincides with the solution of the following quadratic problem (QP) (referred to as the equivalence between SQP and Newton’s method) (Nocedal and Wright, 1999):

$$\begin{align*}
\min_{d} & \frac{1}{2} d' B d + \nabla f(x)' d \\
\text{s.t.} & \quad \nabla G(x)' d = -G(x)
\end{align*}$$

(3.6)

Where $d \in \mathbb{R}^{N_{\text{def}}}$ is the search direction for the current iteration. $B$ in (3.6) may not actually be the Hessian of the Lagrangian (which leads to quadratic convergence), but rather a positive definite approximation.

SQP identifies an optimum of the original NLP by iterating and solving quadratic subproblems of the form of (3.6). The QP problem (3.6) is also generally solved using iterative methods. SQP iterations are called outer or major iterations, whereas QP iterations are called inner or minor iterations.

So far the algorithm derived only considered Equations (3.1) – (3.2) which define an equality-constrained NLP problem. Suppose that the general NLP problem is to be solved:

$$\begin{align*}
\min & \ f(x) \\
\text{s.t.} & \quad G(x) = 0 \\
& \quad h(x) \leq 0 \\
& \quad x^l \leq x \leq x^u
\end{align*}$$

(3.7)

It can be shown that the QP subproblem solved in every (major) iteration of SQP becomes:

$$\begin{align*}
\min_{d} & \frac{1}{2} d' B d + \nabla f(x)' d \\
\text{s.t.} & \quad \nabla G(x)' d = -G(x) \\
& \quad \nabla h(x)' d \leq -h(x) \\
& \quad (x^l - x) \leq d \leq (x^u - x)
\end{align*}$$

(3.8)

The formulation of (3.8) follows the inequality-constrained QP (IQP) approach as presented in Section 2.3. An alternative approach is to adopt the equality-constrained QP formulation (EQP), which was presented in the same section. Both IQP and EQP minimise the same
3. Model reduction-based constrained optimisation of large dissipative systems

objective function, but in the first the Lagrange multipliers are obtained from the QP subproblem whereas in the second are estimated from least squares. Moreover, in IQP an approximation of the full Hessian of the Lagrangian is used while in EQP a projected approximate is utilized. Those differences stem from the fact that in IQP the linearised inequalities of the original NLP are passed on to the QP, while in the EQP an estimation of the active set (i.e. the equality and the active inequality constraints) is used for the formulation of the subproblems which leads to only equality-constrained QP.

3.3 The reduced Hessian SQP methods

The reduced Hessian methods have been reviewed in Section 2.1. They build on the concepts and methodology of Sequential Quadratic Programming, in that they apply quasi-Newton methods to the necessary KKT optimality conditions (Nocedal and Wright, 1999). However they introduce the decomposition of the search space to two subspaces: one being the tangent subspace of the equality constraints and the other being its complement. This way, search directions are computed by component, in a way which will be outlined below. This decomposition enables the elimination of the dependent variables in a systematic and robust way and leads to the formulation of reduced QP subproblems.

Reduced Hessian methods are appropriate for problems involving more dependent variables than the independent ones, typically by one or more orders of magnitude (Schmid and Biegler, 1994). They are also appropriate for problems, the second derivatives of which are computationally expensive or impractical to compute (Nocedal and Wright, 1999). The approximation of the Hessian employed in the formulation is of size $dof \times dof$. This defines the size of the QP subproblem solved per (major) iteration, from which consequently only the independent variable component of the search direction is calculated. Solving the underlying QP is potentially inexpensive if $dof$ is small. Since the Hessian is projected onto the tangent space of the constraints, the chance of it being positive definite (obviously without enforcing this property) is greater than in the standard SQP method (Nocedal and Wright, 1999).
The symbols were explained in the previous Section. If the dimension of the problem is small to medium, the method of choice would be SQP. For large scale systems that include a small number of degrees of freedom, the reduced Hessian methods are more appropriate. The basis of this method is the decomposition the space of the search direction in two subspaces, one being the tangent space of the constraints \( \nabla G(x) \) and the other being its complement. Let \( Z \) and \( Y \) non-orthonormal bases for the two subspaces respectively. The matrix \( [Z \ Y] \in \mathbb{R}^{(N+dof)\times(N+dof)} \) would be non-singular and span the space of the search direction. By construction,

\[
\nabla G(x) Z = 0
\]

An arbitrarily chosen set of bases is:

\[
Z = \begin{bmatrix}
    -\nabla_v G^\top \nabla_y G^\top \\

I
\end{bmatrix}
\]

\[Y = \begin{bmatrix}
I \\
0
\end{bmatrix}\]

Where \( Z \in \mathbb{R}^{(N+dof)\times dof} \) and \( Y \in \mathbb{R}^{(N+dof)\times N} \). Those bases are updated at every iteration, so at iteration \( k \) they should be denoted \( Z_k \) and \( Y_k \). However since there is no risk of confusion, the iteration index is omitted for simplicity. In the analysis that will follow, the index notation will only be used when quantities from different iterations are included in the same expression. It would be advantageous for numerical stability to choose an orthonormal set of bases, so that:

\[
Z^\top Z = I_dof \\
Y^\top Y = I_y \\
Y^\top Z = 0
\]
Where $I_{\dim}$ is the identity matrix, $I_{\dim} \in \mathbb{R}^{\dim \times \dim}$. One way of obtaining matrices $Z$ and $Y$ is by performing QR factorization on the Jacobian of the equality constraints. This has the advantage that produces orthonormal bases but is often excessively computationally costly.

The search direction $d \left( d \in \mathbb{R}^{n_{\text{dof}}} \right)$ of Equation (3.6) can be decomposed as:

$$d = Yp_r + Zp_z \quad (3.14)$$

The $Z$ component of the search direction is computed from the QP subproblem, whereas the $Y$ component computation relies on a Newton-like iteration applied to (3.2). Hence, $p_z$ converges superlinearly if an approximation of the Hessian is used, whereas for $p_r$ the convergence is quadratic (Nocedal and Wright, 1999, p.547). This may lead to what is called tangential convergence, mathematically expressed as:

$$\frac{\|p_r\|}{\|p_z\|} \rightarrow 0 \quad (3.15)$$

This is the reason why some researchers neglect $p_r$ and only compute $p_z$. The benefit from this approximation is the reduced computational cost. However if the bases chosen for the decomposition of the search space are non-orthonormal the range-space component of the search direction may be significant (Biegler et al., 1995).

The linear constraints of the optimisation problem become:

$$G(x) + \nabla G(x)^T Yp_r = 0 \quad (3.16)$$

Therefore assuming that $\nabla G(x)$ has full column rank, since $[Z \quad Y]$ is non-singular, $\nabla G(x)^T Y$ is not degenerate and $p_r$ can be computed as:

$$p_r = -\left( \nabla G(x)^T Y \right)^{-1} G(x) \quad (3.17)$$

At every iteration, a QP subproblem is formulated as follows:

$$\min_{p_z} \left( Z^T f + Z^T B Yp_r \right) \overline{p}_z + \frac{1}{2} p_z^T \left( Z^T BZ \right) p_z$$

s.t. $$(x^f - x) \leq Yp_r + Zp_z \leq (x^u - x) \quad (3.18)$$
Note that (3.18) is reduced and is of dimension \( \text{dof} \). The matrix \( Z^T B Z = B_k \), \( B_k \in \mathbb{R}^{\text{dof} \times \text{dof}} \) is the reduced Hessian and can be computed numerically. A quasi-Newton Broyden – Fletcher – Goldfarb – Shanno (BFGS) update procedure is usually employed:

\[
B_{k+1} = B_k - \frac{B_k s_k y_k^T}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{3.19}
\]

Where \( s_k = \alpha_k p_z \) and \( y_k = Z^T (\nabla_x L(x_{k+1}, \lambda_{k+1}) - \nabla_x L(x_k, \lambda_{k+1})) - \bar{w}_k \), with \( \bar{w}_k = \alpha_k s_{k+1} (y_k p_y) \) an approximation to the null-space step. \( S_k \) is the Broyden matrix, an approximation to \( Z_k^T B_k \) (cf. Biegler et al., 1995).

Estimates of the Lagrange multipliers needed for this calculation can be calculated from:

\[
Y^T B Y p_y + Y^T B Y p_z + Y^T G \lambda = -Y^T \nabla f \tag{3.20}
\]

Since \( B_k \) is computed from a BFGS procedure, it is positive definite; consequently the solution of the QP (3.18) can be computed explicitly:

\[
p_z = -\left( Z^T B Z \right)^{-1} \left( Z^T \nabla f + Z^T B Y p_y \right) \tag{3.21}
\]

This computation is expensive because it involves the “cross-term” \( Z^T B Y p_y \). A more efficient calculation involves approximating this term numerically by a vector, \( w_k \); two approaches can be followed: one being based on Broyden updates and the other on direct calculation through finite differences (cf. Biegler et al., 1995).

Following the calculation of the search direction, the variable vector can be updated:

\[
x_{k+1} = x_k + \alpha_k d_k \tag{3.22}
\]

Where \( \alpha_k \) is a scalar steplength, suitable for reducing the value of a merit function. An example merit function which minimises simultaneously the objective function and the deviation from a feasible point is:

\[
\phi_\mu(x) = f(x) + \mu \| G(x) \|_1 \tag{3.23}
\]

Where \( \mu \) is an appropriate penalty scalar parameter.
The algorithm outlined above follows the simultaneous analysis and design paradigm, so it is an infeasible point algorithm: only the variables at the end of the optimisation procedure are expected to satisfy the constraints.

### 3.4 Model reduction

#### 3.4.1 The reduction approach

The core of the proposed optimisation algorithm is model order reduction (MOR) technology which exploits the separation of time-scales exhibited in dissipative engineering problems. It is expressed as a gap in the eigenspectrum of the Jacobian. A plot of the eigenspectrum of such system is presented in Figure 3.1. The bulk of eigenvalues have algebraically small real part, while a small cluster of eigenvalues with relatively larger real part is distinguishable. Between the two clusters no eigenvalues exist; this is referred to as “spectral gap” and is not necessarily clearly visible. Even in cases where the gap is not so clear visually, the separation of modes to slow and fast may be significant enough for model reduction techniques to build on. Many MOR methods exploit this separation in order to obtain a reduced model. A concise review of such methods has been given in the previous chapter.

The eigenvalues of the Jacobian with the smallest real part (the left-most ones in the spectrum) influence the convergence behaviour of the underlying steady-state simulators (Pashos et al., 2009 and references within). However, it is the right-most eigenvalues that represent the dominant dynamics of the system (Saad, 1992). These are the eigenvalues which determine stability: if all eigenvalues have negative real part, then the underlying system is stable, if at least one has positive real part, then the system is unstable. If any of the eigenvalues have zero real part, the stability of the original nonlinear system cannot be determined from its linear stability (i.e. from the eigenvalues of the Jacobian).
Figure 3.1: Eigenspectrum of a dissipative system exhibiting a clear separation of modes.

A basis for the subspace corresponding to these eigenvalues can be identified at a low computational cost, especially if the cluster is clearly separated from the bulk of eigenvalues (i.e. the spectral gap is clear) and consists of a small number of eigenvalues (Saad, 1992). Iterative methods can be employed for this computation. Krylov subspace methods such as subspace iterations and the Arnoldi method are among the well-formulated and tested iterative methodologies (Lehoucq and Scott, 1996). An advantage of utilization of such methods is their matrix-free nature; they can be applied for the eigenanalysis of the Jacobian even if the latter is not explicitly given. This characteristic enables them to be applied efficiently for large problems, or problems in which the Jacobian is not computed or provided for eigenanalysis.

Let $\mathbf{P}$ be the subspace corresponding to the rightmost eigenvalues and $\mathbf{Q}$ its complement in $\mathbb{R}^n$, so that

$$\mathbf{P} \oplus \mathbf{Q} = \mathbb{R}^n$$

(3.24)
For the class of systems considered in this chapter, \( P \) is heuristically shown to be the dominant subspace of the system, in that it approximates the kernel of the system sufficiently well. Hence, projecting a solution onto \( P \) will not significantly alter it:

\[
\| x - P x \| < \varepsilon, \quad \varepsilon \in \mathbb{R}_+ \quad (3.25)
\]

Let \( \hat{Z} \in \mathbb{R}^{N \times M} \) an orthonormal basis for the dominant subspace \( P \), computed with an iterative procedure as described above. Since it is orthonormal, the following equalities hold (Saad, 1992):

\[
\hat{Z} \hat{Z}^T = P \\
\hat{Z}^T \hat{Z} = I_m 
\]

(3.26)

Where \( P \in \mathbb{R}^{N \times k} \) is the orthogonal projection operator onto \( P \). From (3.24):

\[
Q = I_N - P 
\]

(3.27)

A basis for the subspace \( Q \) would be high-dimensional; however it will not be used in this framework and need not be computed.

Exploiting (3.25), the dependent variables vector, \( u \), can be replaced with its projection, \( v \):

\[
v = \hat{Z}^T u.
\]

Likewise, the Jacobian of the constraints, \( J = (\nabla, G)^T \) can be projected onto \( P \), so that its low-dimensional projection can be used instead of the original matrix. The reduced Jacobian, \( H \in \mathbb{R}^{m \times m} \) is defined as:

\[
H = \hat{Z}^T J \hat{Z} 
\]

(3.28)

A practical and computationally efficient way of computing \( H \), which does not require the explicit computation of \( J \), will be outlined in the next Section.

The dominant subspace \( P \) only refers to the dependent variables and does not include the space of the decision variables. Let \( P_{\text{ext}} \) be the union of the two aforementioned subspaces. An orthonormal basis for \( P_{\text{ext}} \), \( \hat{Z}_{\text{ext}} \in \mathbb{R}^{(N + k \cdot \text{dof}) \times (m \cdot \text{dof})} \) can be computed as:

\[
\hat{Z}_{\text{ext}} = \begin{bmatrix} \hat{Z} & 0 \\ 0 & I_{\text{dof}} \end{bmatrix}
\]

(3.29)
It can be shown that a basis for the subspace of the decision variables, \( Z_r \in \mathcal{R}^{n_{\text{dof}}-\text{dof}} \), can be computed as:

\[
Z_r = \left[ -H^{-1} \dot{Z}^T \nabla_r G^T \right] 
\]  

(3.30)

This basis can be used to project a vector belonging to \( P \) onto the subspace of the decision variables. A basis for the subspace of the independent variables can now be computed by combining \( \hat{Z}_{\text{ext}} \) and \( Z_r \):

\[
Z^* = \hat{Z}_{\text{ext}} Z_r = \left[ \hat{Z} \quad 0 \right] \left[ \begin{array}{c} -H^{-1} \dot{Z}^T \nabla_r G^T \\ 0 \end{array} \right] = \left[ \begin{array}{c} -\hat{Z} H^{-1} \dot{Z}^T \nabla_r G^T \\ 0 \end{array} \right] 
\]  

(3.31)

The newly computed basis \( Z^* \in \mathcal{R}^{n_{\text{dof}}-\text{dof}} \) constitutes a new coordinate basis for the subspace of the independent variables and it can be seen as the counterpart of \( Z \) defined by (3.11) in the reduced Hessian framework. The main difference is the incorporation of the model reduction step, which leads to a computation only including low-dimensional quantities. The (possibly overwhelming) hurdle of inverting the Jacobian in order to compute this basis is leaped. The bases \( Z^* \) and \( Z \), although they approximate the same quantity are not equal.

Following this rationale, all the quantities involved in the formulation of the proposed optimisation algorithm will undergo a projection onto the dominant subspace of the system. For most of them, this will be followed by another one, onto the subspace of the decision variables. The reduced Hessian, \( \hat{B}_r \in \mathcal{R}^{n_{\text{dof}}-\text{dof}} \), also undergoes the two-step projection scheme described:

\[
\hat{B}_r = Z_r^T \hat{Z}_{\text{ext}} B \hat{Z}_{\text{ext}} Z_r = Z^{**} B Z^*
\]

(3.32)

In practice, the basis \( Z^* \) will be used for the efficient computation of \( \hat{B}_r \) from (3.32) by performing numerical directional perturbations directly on the Lagrangian function of the system. In this computation, only low-dimensional projections of the Lagrange multipliers onto the dominant subspace will be used:

\[
\phi = \hat{Z}^T \lambda
\]

(3.33)

Computed from:
3. Model reduction-based constrained optimisation of large dissipative systems

\[ H^T \phi = -\hat{Z}^T \nabla f \tag{3.34} \]

### 3.4.2 The effect of model reduction in the optimisation

In this section it is shown that every optimum of the original NLP problem is also an optimum of the reduced (projected) problem as well.

Another way of writing the first-order KKT conditions of the full problem (Equation (3.4)), is:

\[ (\nabla f)^T + (\nabla G)^T \lambda = 0 \tag{3.35} \]

And

\[ G = 0 \tag{3.36} \]

Recall that \( P_{ext} \) is the extended dominant subspace to include the space of decision variables, defined by a projector \( P_{ext} \) and spanned by the orthonormal basis \( \hat{Z}_{ext} \). \( P_{ext} \) is an orthogonal projector. Corollary 3.1 is easy to show with the aid of Equations (3.26) and (3.29).

**Corollary 3.1**: The projector \( P_{ext} \) satisfies the equalities of Equation (3.37):

\[
\begin{align*}
P_{ext}^2 &= P_{ext} \\
P_{ext} &= \hat{Z}_{ext} \hat{Z}_{ext}^T \\
\hat{Z}_{ext}^T \hat{Z}_{ext} &= I_{N_{\text{dof}}} 
\end{align*}
\tag{3.37}
\]

What is effectively solved with the proposed algorithm is the projection of the original NLP on the subspace \( P_{ext} \) using a reduced Hessian method. Thus, the first-order KKT conditions which are solved in the proposed algorithm are projections of conditions (3.35) and (3.36) onto \( P_{ext} \).

The necessary KKT conditions of the reduced problem are:

\[ \hat{Z}_{ext} (\nabla f)^T + H_{ext}^T \phi = 0 \tag{3.38} \]

And
Theorem 3.1: For every point satisfying (3.35), (3.38) will also hold.

**Proof:** Project (3.35) onto $P_{\text{ext}}$:

$$P_{\text{ext}} (\nabla f)^T + P_{\text{ext}} \begin{bmatrix} \nabla G^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ 0 \end{bmatrix} = 0$$

And then restrict it to this subspace by multiplying it from the right with $P_{\text{ext}}$:

$$P_{\text{ext}} (\hat{f}^T \nabla f + \hat{f}^T \nabla G^T 0) \hat{\lambda}_0 = 0$$

Multiplying with $\hat{Z}_{\text{ext}}^T$:

$$\hat{Z}_{\text{ext}}^T (\nabla f)^T + \hat{Z}_{\text{ext}}^T \begin{bmatrix} \nabla G^T & 0 \end{bmatrix} \hat{\lambda}_0 = 0$$

Since $\hat{Z}_{\text{ext}}^T \hat{Z}_{\text{ext}} = I_{N \times N}$ (Eq. (3.37)):

$$\hat{Z}_{\text{ext}}^T (\nabla f)^T + \hat{Z}_{\text{ext}}^T \begin{bmatrix} \nabla G^T & 0 \end{bmatrix} \hat{\lambda}_0 = 0$$

The second KKT condition (Equation (3.40)) is easy to show and will be referred to as Corollary 3.2:

**Corollary 3.2:** If (3.36) holds, then $P_{\text{ext}} \begin{bmatrix} G \\ 0 \end{bmatrix} = 0$. 
Theorem 3.1 and Corollary 3.2 can be used to prove Corollary 3.3, which is of central importance to the justification of the proposed optimisation algorithm.

**Corollary 3.3**: Every point which satisfies the first-order KKT conditions of the full problem (Equations (3.35) and (3.36)) satisfies those conditions for the reduced problem as well (Equations (3.38) and (3.39)).

Recall that with SQP and rSQP, extrema are located by applying (quasi-)Newton’s method on those conditions. Therefore, Corollary 3.3 actually guarantees that by applying the proposed optimisation scheme, convergence to the optimum points of the original points is possible. This of course does not guarantee that the reduced problem does not exhibit additional stationary points which satisfy the necessary KKT conditions.

Assume that the proposed algorithm converges to a point \( x^* \). At this point, the projection of the necessary KKT conditions onto \( P \) are satisfied:

\[
P\nabla f + P\nabla G = 0 \quad \text{and} \quad PG = 0 \quad (3.41)
\]

In order to satisfy the KKT conditions of the full problem (Equations (3.4)), the projections onto \( Q \) also have to hold:

\[
Q\nabla f + Q\nabla G = 0 \quad \text{and} \quad QG = 0 \quad (3.42)
\]

This is a consequence of Equation (3.24). Point \( x^* \) is feasible, since this is a feasible-path algorithm, hence \( G = 0 \), therefore \( QG = 0 \). Thus the only possible scenario for \( x^* \) not to satisfy the necessary conditions (3.35) and (3.36) (and therefore qualify as an optimum for the full problem), is for \( Q\nabla f + Q\nabla G = 0 \) and \( QG = 0 \) to simultaneously hold.

### 3.5 The proposed optimisation algorithm

The proposed optimisation scheme results from assembling the concepts outlined above. Initialisation requires initial guesses for the solution vector (used in the steady-state simulator) and the reduced Hessian, which should be positive definite and is used in lieu of
an estimate of a reduced Hessian in the first iteration of the algorithm. Aspects of the algorithm will be explored in Sections 3.5.1 – 3.5.5.

3.5.1 Using an existing iterative solver for the computation of a feasible point

The proposed algorithm belongs to the reduced space SQP class of methods. Usually such methods are implemented in infeasible point algorithms, in which the intermediate points of the optimisation procedure do not satisfy the constraints. This leads to “simultaneous analysis and design” (SAND) procedures, in which the simulation problem is solved simultaneously with the optimisation one. Nevertheless, this work adopts the rival “nested analysis and design” (NAND) approach, as it is assumed that an appropriate solver for converging the steady-state problem already exists and the optimiser will need to form a superstructure (“wrapper”) around it. This approach has the advantage that previous modelling work is exploited to the full. The simulator could potentially be a result of many person-years of work; this could especially be the case if the underlying physical system is stiff and requires heuristics or other special methods to be solved. In these systems employing Newton-like procedures, like in the SAND approach, may fail. Another advantage of the approach followed is that it allows to safely ignore the range-space component of the search direction, $p_r$, the computation of which is expensive. On the other hand, an obvious disadvantage of this approach is that converging to a feasible point may prove to be costly; for such cases allowing the black-box simulator to perform a certain maximum number of convergence steps may be preferred.

The steady-state simulator is inputted the tentative solution vector $x$, resulting from following the search direction provided by the QP subproblem and replaces it with a new one which satisfies the equality constraints: $G(x)=0$. In the simplest case, this simulator consists of a home-made code implementing Newton’s method. New iterates of the $k$-th iteration of this algorithm are computed from the following equation:

$$u^{k+1} = u^k - J(x^k)^{-1}G(x^k) \quad (3.43)$$

Notice that in Equation (3.43) the decision variables are treated as parameters and remain unaltered. However, since Newton’s method is only locally convergent and it is possible for the optimisation procedure to result in significant jumps in search space, iteration (3.43) may fail due to the stiffness of the underlying problem. This is likely, since $u^0$ satisfies the
constraints for different values of the decision variables and therefore may prove to
comprise a poor initial guess for the current \( z \). Iteration (3.43) can be relaxed, so that the
full variable vector (including dependent and independent variables) is updated from a
Newton-like iteration using an augmented Jacobian, \( J_m \):

\[
(J_m(x^k)^T J_m(x^k))x^{k+1} = x^k - G(x^k)
\]  

(3.44)

With \( J_m \) being computed from:

\[
J_m = \begin{bmatrix}
\nabla_x G(x)^T \\
\nabla_z G(x)
\end{bmatrix}
\]  

(3.45)

Note that \( J_m \) as defined in Equation (3.45) is not a square matrix, so its inverse does not
exist. For this reason, the pseudo-inverse of \( J_m \) is used in iteration (3.44).

### 3.5.2 Model reduction and computation of the reduced quantities

The proposed algorithm circumvents the need for the computation and inversion of the full
Jacobian matrix, as required both in SQP and in rSQP methods. It is common practice to
avoid computations of full Jacobians. In simulation software, approximate analytical
Jacobians are usually formed by neglecting terms (such as source terms) and typically
matrix-free methodologies are employed which deal with Jacobian-vector products and
thus avoid explicitly forming the full matrix. In this work, only low-dimensional Jacobians
are used, as per Equation (3.28), which are efficiently computed projections of the original
Jacobian onto the dominant subspace of the system. This computation is performed
numerically with directional perturbations and requires a basis for this dominant subspace.

The algorithm outlined in this Section revolves around the idea that there exists a subspace
which is termed dominant, corresponding to the eigenvalues of the Jacobian with the
algebraically largest real part and approximates the null-space of the system sufficiently. A
basis for this subspace can be computed efficiently by utilizing matrix-free techniques such as
subspace iterations or Arnoldi method. The number \( m \) of eigenvalues of the Jacobian
considered to consist the dominant cluster of modes is usually set to a constant, but
adaptive identification of the spectral gap and consequently \( m \) is possible, by monitoring
the convergence rate of the matrix-free procedure that computes the basis of the
eigenspace belonging to those modes. Adaptive estimation of $m$ leads to the maximal invariant subspace belonging to the dominant modes. The iterative methods used to identify the eigenspace are more efficient when used to identify clusters of eigenvalues and in this case the existence of spectral gap is advantageous. In general, the number of dominant modes varies depending on the region in the parameter space. Some simulators are able to provide information about the eigenspectrum of the underlying system. If that is the case, this option should be taken. In any case, any subspace merely containing the dominant subspace can be used as $P$. Overestimating the number of dominant modes by setting the value of $m$ is the conservative choice. The size of the dominant subspace is a characteristic of the physical system and does not depend on the type of the simulator or on discretisation scheme and number of nodes.

The matrix-free methods suggested for the computation of a basis spanning the dominant of the system do not require the Jacobian been given explicitly, since provision of a procedure which returns the product of the Jacobian with given vectors is adequate. In this work subspace iterations were performed using subroutine EB22 of the HSL library (HSL, 2007). The (vector) product of the Jacobian of the equality constraints, $J(x)$, with a given vector, $v$ following the numerical directional perturbation paradigm and the central Finite Difference scheme is computed as:

$$J(x)v = \frac{1}{2 \varepsilon_j} (G(x + \varepsilon_j v) - G(x - \varepsilon_j v)), \quad \varepsilon_j \in \mathbb{R},$$  \hspace{1cm} (3.46)$$

A less expensive scheme (e.g. forward differences) can lead to significant reduction of function evaluations but this reduction may be to the expense of computational robustness. Such matrix-free techniques are widely used in computational methods for large-scale systems, as matrices are often large and evading their computation leads to significant savings.

The procedure for computing reduced Jacobians has roots in the Recursive Projection Method (Shroff and Keller, 1993). Recall that the reduced Jacobian is defined in Equation (3.28). It is formed in two steps. First the $J\hat{Z}$ product is computed and then premultiplied by $\hat{Z}^T$. The basis $\hat{Z}$ can be seen as a collection of $m$ vectors. The column $j$ of the matrix $J\hat{Z}$ can be computed from Equation (3.46), where the vector $v$ is the $j$-th column of $\hat{Z}$.
3.5.3 Computation of the reduced Hessian and the Lagrange multipliers

After the model reduction step, a basis for the subspace of the decision variables needs to be computed. It is formed following a two-step projection scheme. The first projection is onto the dominant subspace of the system and has already been outlined. A basis \( \hat{Z} \) for this subspace is identified with subspace iterations and is then extended to include the subspace of the degrees of freedom and enable the second projection. A basis for the extended subspace is \( Z_{\text{ext}} \) can be computed as per Equation (3.29).

The second projection is onto the tangent space of the dominant modes of the system, which yields the subspace of the independent variables. A basis for the second projection is \( Z_r \) and is computed as per Equation (3.30). A basis for the final subspace to which one projects following the two-step scheme is \( Z^* \) and is computed as per Equation (3.31).

The reduced Hessian is projected with the aid of \( Z^* \) as:

\[
B_h = Z^T B Z^* = Z^T (\nabla^2 L(\chi, \lambda))^T Z^*
\]  

(3.47)

Like the reduced Jacobian, the reduced Hessian is most efficiently calculated with numerical directional perturbations. Here, it is the Lagrange function of the projected problem that is being perturbed. This function is defined as:

\[
L(\chi, \lambda) = f(\chi) + \phi^T \hat{Z}^i G(\chi)
\]  

(3.48)

Where \( \chi \in \mathbb{R}^{m+\text{dof}} \) is the projected variable vector: \( \chi = \hat{Z}_{\text{ext}} \chi \). Note that only \( m \) Lagrange multipliers are used in the reduced Lagrange function. The reduced Lagrange multiplier vector, \( \phi \) is defined by Equation (3.33) and is computed from Equation (3.34). The Lagrange function can either be computed as \( f(\chi) + (\hat{Z} \phi)^T G(\chi) \), or as \( f(\chi) + \phi^T (\hat{Z}^i G(\chi)) \). So the reduced Lagrange multipliers can either be projected back onto the full \( N \)-dimensional space and used directly in conjunction with the residual vector, or alternatively it is the latter vector that will be projected onto the dominant \( m \)-dimensional space and used with the reduced multipliers.

The reduced Hessian is a \((\text{dof} \times \text{dof})\) matrix. The element \((i, j)\) is computed from the second-order finite difference formula:
3. Model reduction-based constrained optimisation of large dissipative systems

\[ B_{n,i} = \frac{L(x + \varepsilon Z_i' + \varepsilon Z_j') + L(x - \varepsilon Z_i' + \varepsilon Z_j') + L(x + \varepsilon Z_i' - \varepsilon Z_j') + L(x - \varepsilon Z_i' - \varepsilon Z_j')}{4\varepsilon^2} \]  

(3.49)

with \( \varepsilon > 0 \) the perturbation size. Note that since the Hessian refers to second derivative, whereas the Jacobian to first, the \( \varepsilon \) in (3.49) is chosen to be to the order of the square root of the \( \varepsilon_j \) of Equation (3.46): \( O(\varepsilon) = O\left(\sqrt{\varepsilon_j}\right) \).

The Hessian matrix of any function is by definition symmetric. Element \((i, j)\) represents \( \frac{\partial^2 L(x)}{\partial x_i \partial x_j} \) whereas the element \((j, i)\) represents \( \frac{\partial^2 L(x)}{\partial x_j \partial x_i} \). The computed from Equation (3.49) reduced Hessian may not be symmetric due to round-off errors. Nevertheless, it can easily be forced to be symmetric by defining:

\[ B_n \leftarrow \frac{1}{2}(B_n + B_n^T) \]  

(3.50)

The reduced Hessian calculated from (3.49) and (3.50) will only be semipositive definite in the neighbourhood of the optimum for non-convex functions, therefore the quadratic subproblem solved at every iteration is not guaranteed to be feasible:

\[
\min_{p_x} \left( Z^T \nabla f_z \right)^T p_x + 1/2 p_x^T B_n p_x, \quad \text{s.t.} \quad (x^i - x) \leq Z^T p_x \leq (x^{ii} - x) \]

(3.51)

Global convergence, i.e. convergence to the optimum irrespective of the choice of the initial guess, can be achieved using an approximation procedure such as the BFGS of Equation (3.19), which will maintain the positive definiteness of \( B_n \) throughout the updates. However it will reduce the quality of the Hessian (already somewhat compromised due to model reduction) and cause convergence to deteriorate.

From numerical experience with the proposed algorithm it was observed that using reduced Hessians computed from Equations (3.49) and (3.50), results in convergence in less iterations. However implementing a “failsafe” system is advisable. Namely if a QP fails to converge then the computation is repeated by performing BFGS updates instead of exact calculation of the Hessians.
3.5.4 Updating the solution

The solution vector is updated in each iteration using the search direction computed from the QP subproblem:

\[ x_{k+1} = x_k + \alpha_k Z^* p_z \]  \hspace{1cm} (3.52)

A unit steplength may be chosen, \( \alpha_k = 1 \), so that (3.52) becomes: \( x_{k+1} = x_k + Z^* p_z \).

Alternatively, a value for the steplength can be computed using linesearch. For example following an Arnijo linesearch procedure, \( \alpha_k \) will be calculated as (Edgar et al., 2001):

\[ \alpha_k = \frac{\nabla^T G(x_k) Z^* p_z}{2 \left( G(x_k + Z^* p_z) - G(x_k) - \nabla^T G(x_k) Z^* p_z \right)} \]  \hspace{1cm} (3.53)

Employing linesearches is a popular globalization strategy. It is optional and could result in poor convergence near the optimum, a phenomenon which is called Maratos effect.

3.5.5 Convergence monitoring and termination

The standard termination criterion of SQP is employed in the proposed scheme (cf. Edgar et al., 2001). The optimisation procedure stops if:

\[ \frac{\| x_k - x_{k-1} \|}{\| x_k \| + 1} < \epsilon_1 \]  \hspace{1cm} (3.54)

For a given tolerance \( \epsilon_1 > 0 \). For infeasible path implementations, an additional termination criterion is added:

\[ \| G(x_k) \| < \epsilon_2 \]  \hspace{1cm} (3.55)

For a given tolerance \( \epsilon_2 > 0 \). Convergence monitoring is performed using the quantities \( \| Z^T \nabla f \| \) and \( \| Z^* p_z \| \) which have been shown to be indicative of the convergence procedure (Biegler et al., 1995).
Table 3.1: Algorithm 3.1: The proposed optimisation algorithm

1. Choose initial guesses for the variables $x_0$ and the Hessian $B_0$

2. Calculate a steady state using an iterative solver and evaluate $x$ and $f$

3. Model reduction step: identify a basis for the dominant subspace, $\hat{Z}$, with subspace iterations and exploit it for the computation of the reduced Jacobian, $H$ from

   $H = \hat{Z}'\hat{Z}$  \hspace{1cm} (18)

   with numerical directional perturbations

4. Calculate the basis for the subspace of the decision variables, $Z'$:

   $Z' = \hat{Z}_{\text{opt}}Z' = \begin{bmatrix} \hat{Z} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} -H'\hat{Z}'\nabla_x G' \\ I \end{bmatrix} = \begin{bmatrix} -\hat{Z}H'\hat{Z}'\nabla_x G' \\ I \end{bmatrix}$

5. Calculate the reduced Hessian, $B'_R = Z'^TBZ'$ by performing numerical perturbations on the Lagrange function using $Z'$ and $\phi$

6. Solve the QP subproblem:

   $\min_{p_z} (Z'^\nabla f_z) p_z + 1/2 p_z' B_R p_z,$  \hspace{1cm} s.t.  \hspace{1cm} $(x^i - x) \leq Z'^* p_z \leq (x^u - x)$

7. Update the solution: $x = x + Z'p_z$

8. Update $\hat{Z}$ and calculate the corresponding $H$ as in step (3)

9. Compute the reduced Lagrange multipliers, $\phi$: $H'\phi = -\hat{Z}'Y'\nabla f$

10. Check for convergence: $\frac{\|x_k - x_{k-1}\|}{\|x_k\| + 1} < \epsilon_v$. If convergence is not achieved return to step 2.
3.6 A computationally efficient implementation

The most expensive step of the algorithm outlined in Section 3.5 is the computation of the bases for the dominant subspace. The basis $\hat{Z}$ is updated twice in Algorithm 3.1: in steps (3) and (8). This update is the most expensive step of the algorithm, especially if the two points at which $\hat{Z}$ is calculated differ significantly. In this Section two techniques of reducing this cost are presented.

The first approach is to “warm-start” the subspace iterations (or the equivalent methodology which is employed for the update of $\hat{Z}$), by providing a good initial guess for the desired basis. Providing no initial guess for $\hat{Z}$ is called cold-starting. In that case, an random matrix is generated and used in lieu of an initial guess. After the first basis has been computed, one may use the last computed $\hat{Z}$ as an initial guess for the update. In the first iteration, when no basis has been computed, one can use the sequence of the residuals of the steady-state simulator used in step (2) of Algorithm 3.1. Those residuals converge to the dominant eigenspace of the system (Shroff and Keller, 1993). Although Newton-like simulators may exhibit abrupt convergence, warm starting may prove to accelerate convergence over the alternative of using a random matrix. In case the solver converges in less iterations than the number of vectors required for warm-starting subspace iterations, some columns of the initial guess will remain nil. One can either use extrapolation methods, e.g. with rational polynomials, to extrapolate the residuals versus the iteration number. This approach is simplistic but could provide a better alternative than cold-starting. Another possible way is to employ Arnoldi or the power method, which only starts from one vector (this case that would be the last residual), in order to obtain a number of vectors which can be provided as initial guess to subspace iterations.

The second approach proposed is more drastic than the first one. An implementation of Algorithm 3.1 is introduced, where updating of the basis is only performed once per iteration. This is implemented in Algorithm 3.2. The underlying assumption is that the basis of the dominant subspace computed for the values of independent variables directed from the QP subproblem, can be approximated with the basis at the corresponding feasible point identified using the simulator.

By making this assumption, one can calculate estimation of the reduced Lagrange multipliers of the next iteration at the next iteration. The computation is based on Equation
as before, but although the symbols $\nabla f$ and $x$ refer to the point computed from the QP subproblem, $\hat{z}$ refers to the one computed from step 3. So for iteration $k+1$, Equation (3.34) can be rewritten:

$$H^T \phi_{j+1} = -\hat{z}_{j+1}^T \nabla f(x_j)$$  \hspace{1cm} (3.56)

With

$$H = \hat{z}_{j+1}^T (\nabla_x G(x_j))^T \hat{z}_j$$  \hspace{1cm} (3.57)

The assumption made does incur loss of accuracy but it also it does reduce significantly the computational cost. It is suggested to monitor the norm of the search direction and determine when the jump is minor; if that is the case, the procedure outlined in Algorithm 3.1 is used, otherwise updating of the $\hat{z}$ after the QP is skipped. It will be shown in the next Section that the cost of updating the basis reduces significantly when the algorithm converges, so this rationale reduces the computational cost significantly without compromising at all the accuracy to which convergence to the optimum is achieved. Furthermore, the savings increase more with problem size.

**Table 3.2: Algorithm 3.2:** An efficient implementation or the proposed algorithm

1. Choose initial guesses for the variables $x_0$ and the Hessian $B_0$

2. Calculate a steady state using an iterative solver and evaluate $x$ and $f$

3. Model reduction step: identify a basis for the dominant subspace, $\hat{Z}$, with subspace iterations and exploit it for the computation of the reduced Jacobian, $H$ from $H = \hat{z}_j^T J_{\hat{z}_j}$ (18) with numerical directional perturbations

4. Calculate the basis for the subspace of the decision variables, $Z^*$:

$$Z^* = \hat{Z}_{ext} Z_c = \begin{bmatrix} \hat{z} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} -H^T \hat{z}_j \nabla_x G^T \end{bmatrix} = \begin{bmatrix} -H^T \hat{z}_j \nabla_x G^T \\ I \end{bmatrix}$$

5. If (iteration $> 1$) and ($\|Z^* p_z\| > \varepsilon$) then calculate the reduced Lagrange multipliers, $\phi$, from $H^T \phi = -\hat{z}_j^T \nabla f$ using $x_0$ and $H_0$. 
6. Calculate the reduced Hessian, $B_r = Z^T B Z^*$ by performing numerical perturbations on the Lagrange function using $Z^*$ and $\phi$

7. Solve the QP subproblem:

$$\min_{\mathbf{p}_z} \left( Z^T \nabla f_z \right)^T \mathbf{p}_z + 1/2 \mathbf{p}_z^T B_z \mathbf{p}_z, \text{ s.t. } (x^i - x) \leq Z^* \mathbf{p}_z \leq (x^u - x)$$

8. Update the solution: $x = x + Z^* \mathbf{p}_z$ and set $x_0 = x$

9. If $\|Z^* \mathbf{p}_z\| < \epsilon$ update $\hat{Z}$, calculate the corresponding $H$ and the reduced Lagrange multipliers, $\phi$ as in step 5

10. Set $H_0 = H$

11. Check for convergence. If convergence is not achieved return to step (2)

---

3.7 Case study: Optimisation of a tubular reactor

3.7.1 The base case

This Section presents a series of case studies in order to illustrate the behaviour of the proposed algorithms. The base scenario is the optimisation of a tubular reactor where an exothermic, first-order irreversible reaction takes place: $A \rightarrow B$, as shown in Fig. 3.2.

![Figure 3.2: Schematic representation of a tubular reactor](image_url)
The model of the reactor consists of 2 PDEs which in dimensionless form are (Jensen and Ray, 1982):

\[
\begin{align*}
\frac{\partial x_1}{\partial t} &= \frac{1}{Pe_1} \frac{\partial^2 x_1}{\partial y^2} - \frac{\partial x_1}{\partial y} + Da(1-x_1)\exp\left(\frac{x_2}{1+x_2/\gamma}\right) \\
\frac{\partial x_2}{\partial t} &= \frac{1}{LePe_2} \frac{\partial^2 x_2}{\partial y^2} - \frac{1}{Le} \frac{\partial x_2}{\partial y} - \frac{\beta}{Le} x_2 + \frac{C}{Le} Da(1-x_1)\exp\left(\frac{x_2}{1+x_2/\gamma}\right) + \frac{\beta x_{2w}}{Le}
\end{align*}
\]  

(3.58)

Where \(\gamma\) is the dimensionless heat transfer coefficient, \(C\) is the dimensionless adiabatic temperature rise, \(x_{2w}\) the dimensionless adiabatic wall temperature, \(y\) the dimensionless activation energy and \(y\) the dimensionless longitudinal coordinate.

The boundary conditions for the above set of nonlinear equations are:

\[
\frac{\partial x_1}{\partial y} = 0, \quad \frac{\partial x_2}{\partial y} = 0 \quad \text{at} \quad y = 0
\]  

(3.59)

and

\[
\frac{\partial x_1}{\partial y} = 0, \quad \frac{\partial x_2}{\partial y} = 0 \quad \text{at} \quad y = 1
\]  

(3.60)

The values of the parameters chosen are \(Le = 1.0, Pe_1 = Pe_2 = 5.0, \gamma = 20.0, \beta = 1.50, C = 12.0, x_{2w} = 0.0\).

At steady-state, Equation (3.58) becomes:

\[
\begin{align*}
\frac{1}{Pe_1} \frac{\partial^2 x_1}{\partial y^2} - \frac{\partial x_1}{\partial y} + Da(1-x_1)\exp\left(\frac{x_2}{1+x_2/\gamma}\right) &= G_1 = 0 \\
\frac{1}{LePe_2} \frac{\partial^2 x_2}{\partial y^2} - \frac{1}{Le} \frac{\partial x_2}{\partial y} - \frac{\beta}{Le} x_2 + \frac{C}{Le} Da(1-x_1)\exp\left(\frac{x_2}{1+x_2/\gamma}\right) + \frac{\beta x_{2w}}{Le} &= G_2 = 0
\end{align*}
\]  

(3.61)

The set of equations (3.61) is discretised with the central Finite Differences method on a mesh of 250 nodes. This discretisation results in a 500 dependent variables.

This system exhibits a rich interesting parametric behaviour. In Figure 3.3 the bifurcation diagrams for the two dimensionless variables with respect to the Damköhler number are
presented. Solid lines represent stable branches and broken lines represent unstable ones. The turning points appear at \( Da \approx 0.122 \) and \( Da \approx 0.110 \) and the Hopf bifurcation point at \( Da \approx 0.1100 \) (Koronaki et al., 2003).

### 3.7.2 A 1-dof case

The first case study presented has one independent variable. The objective is to maximize the output dimensionless temperature from the reactor with respect to the Damköhler number:

\[
\max_{Da} x_j \left|_{\text{exit}} \right.
\]

s.t. \( G_{\text{i}|k} = 0 \), \( G_{\text{u}|k} = 0 \),

\[0 \leq x_{\text{i}|k} \leq 1, \quad 0 \leq x_{\text{u}|k} \leq 8\]  

(3.62)

Where \( k \) represents the node number, so in this case \( k \in \{1, 2, \ldots, 250\} \).

The size of the subspace considered is chosen to be large enough to capture the dominant dynamics throughout the parameter space and in this case it is \( m = 10 \).

Convergence is achieved in 8 iterations. The optimisation path is depicted in Figure 3.4, along with the corresponding path for a feasible point implementation of a reduced Hessian method. These paths are different but ultimately converge to the same point. Figure 3.4 can be considered as a visual validation of the proposed method. The computational cost of this procedure as well as comparison with an rSQP and a dense SQP method is presented in Section 3.7.4. Note that apart from the decrease in the computational cost, the reduction in memory requirements is also significant.

Convergence results are given in terms of \( \|z^T \nabla f\| \) and \( \|z^T p\| \) as described in Section 3.6 and are presented in Table 3.3. Using this data, a convergence curve is plotted in Figure 3.5. The linear segment of the convergence curve means fast convergence in the neighbourhood of the optimum. The concentration and temperature profiles for the optimal Damköhler number are depicted in Figure 3.6.
Figure 3.3: Bifurcation diagram dimensionless concentration $x_1$ (a) and temperature $x_2$ (b) of the reactor.
Figure 3.4: The convergence paths for the proposed algorithm in red and a feasible point implementation of the reduced Hessian in green. (●) correspond to QP steps and (□) correspond to Newton steps.

Figure 3.5: Convergence curve for the 1 dof case study
### Table 3.3: Convergence data for the 1 dof case study

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$Da$</th>
<th>$f$</th>
<th>$|Z^TVf|$</th>
<th>$|Z^TP|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.200000</td>
<td>3.60184</td>
<td>2.60E+02</td>
<td>1.80E+02</td>
</tr>
<tr>
<td>2</td>
<td>0.122452</td>
<td>7.19461</td>
<td>46.5400</td>
<td>0.666214</td>
</tr>
<tr>
<td>3</td>
<td>0.114377</td>
<td>6.06487</td>
<td>9.67305</td>
<td>1.914227</td>
</tr>
<tr>
<td>4</td>
<td>0.113842</td>
<td>6.05742</td>
<td>0.236306</td>
<td>0.942793</td>
</tr>
<tr>
<td>5</td>
<td>0.113873</td>
<td>6.05493</td>
<td>6.80E-03</td>
<td>4.73E-02</td>
</tr>
<tr>
<td>6</td>
<td>0.113872</td>
<td>6.05492</td>
<td>1.06E-02</td>
<td>1.08E-03</td>
</tr>
<tr>
<td>7</td>
<td>0.113871</td>
<td>6.05492</td>
<td>2.58E-03</td>
<td>1.22E-03</td>
</tr>
<tr>
<td>8</td>
<td>0.113871</td>
<td>6.05492</td>
<td>2.96E-04</td>
<td>8.51E-05</td>
</tr>
</tbody>
</table>

If instead of using exact second derivatives one chooses to update the reduced Hessian using the BFGS scheme, the number of iterations increases to 13. Implementing an Armijo linesearch for choosing the steplength used for the update in Equation (3.52) does not alter the number of iterations required to converge. Both methods are referred to as globalization strategies, they extend the range of convergence to a local optimum, eliminating the limitation that a bad initial guess would impose. Note the increase of the number of iterations when BFGS is employed, the reasons for which are outlined in Section 3.5.3.

The same case study has also been solved using the modified, efficient, version of the optimisation method as presented in Algorithm 3.2. CPU time is reduced by 10% in comparison to Algorithm 3.1. Let it be noted that this reduction in computational cost increases with problem size. Figure 3.7 presents the reduction of the time taken for the update of the basis for the dominant subspace per iteration, justifying the rationale for
Algorithm 3.2. The convergence data for this case are presented in Table 3.4 and the corresponding convergence curve is plotted in Figure 3.8.

![Graph](image1)

(a)

![Graph](image2)

(b)

**Figure 3.6:** Profiles for the dimensionless concentration (a) and temperature (b) at the optimal $Da$. 
3. Model reduction-based constrained optimisation of large dissipative systems

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$Da$</th>
<th>$f$</th>
<th>$|Z^\top \nabla f|$</th>
<th>$|Z^\top p_i|$</th>
<th>Z-updates per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.200000</td>
<td>1.08741</td>
<td>25.14432</td>
<td>25.7635</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.122451</td>
<td>2.65259</td>
<td>8.78E+03</td>
<td>36.6290</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0.114116</td>
<td>6.05142</td>
<td>11.8029</td>
<td>1.51089</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0.113866</td>
<td>6.05547</td>
<td>0.105973</td>
<td>0.439302</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0.113873</td>
<td>6.05492</td>
<td>5.60E-03</td>
<td>1.43E-02</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0.113873</td>
<td>6.05492</td>
<td>1.07E-02</td>
<td>4.24E-04</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>0.113873</td>
<td>6.05492</td>
<td>3.43E-04</td>
<td>3.76E-04</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>0.113873</td>
<td>6.05492</td>
<td>2.43E-04</td>
<td>3.55E-04</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0.113873</td>
<td>6.05492</td>
<td>5.12E-05</td>
<td>1.92E-05</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 3.4:** Convergence data for the 1 dof case study using Algorithm 3.2
3. Model reduction-based constrained optimisation of large dissipative systems

**Figure 3.7:** Reduction of the time needed for the update of the basis $\hat{Z}$ per iteration for the 1-dof case study.

**Figure 3.8:** Convergence curve for the 1 dof case study using Algorithm 3.2.
3.7.3 A 3-dof case study

A second case study has also been formulated based on the tubular reactor base case. Three cooling zones are considered in the jacket of the reactor, the temperature of which can be set independently. The decision variables are the dimensionless temperatures of the cooling zones, \( x_{2w_i} \), which vary spatially:

\[
x_{2w_i}(y) = \sum_{j=1}^{3} \left( H(y - y_{j-1}) - H(y) - y_j \right) x_j(t)
\]  
(3.63)

With \( y_i = i/3, i = 0, 1, ..., 3, x_{2w_j}, j = 1, 2, ..., 3, \) is the dimensionless temperature of the cooling zone \( H(\cdot) \) is the Heaviside step function:

\[
H(y) = \begin{cases} 
0, & y < 0 \\
1, & y \geq 0 
\end{cases}
\]  
(3.64)

The Damköhler number is set to 0.1. The objective is the maximization of the output dimensionless concentration at the exit of the reactor:

\[
\max_{x_{\text{exit}}} \left. x_k \right|_{\text{exit}} \\
\text{s.t. } G_1 = 0, \quad G_2 = 0, \\
0 \leq x_k \leq 1, \quad 0 \leq x_k \leq 8
\]  
(3.65)

Where \( k \) represents the node number: \( k \in \{1,2,\ldots,250\} \) and \( i \) refers to the cooling zone, \( i \in \{1,2,3\} \).

Convergence in this case is achieved in 10 iterations. Convergence results are given in terms of \( \|z^T \nabla f\| \) and \( \|z^T p\| \) as before and are recapitulated in Table 3.5. The corresponding convergence curve is plotted in Figure 3.9. As in the previous case, a high convergence rate close to the optimum is observed. The concentration and temperature profiles both for reactor and for the jackets at the identified optimum are plotted in Figure 3.10.
### Figure 3.9: Convergence curve for the 3 dof case study

### Table 3.5: Convergence data for the 3dof case study using Algorithm 3.1

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$x_{2w}$</th>
<th>$x_{2w}$</th>
<th>$x_{2w}$</th>
<th>$f$</th>
<th>$|z^T \nabla f|$</th>
<th>$|z^T p_z|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.001723</td>
<td>1.999998</td>
<td>2.000082</td>
<td>0.99861</td>
<td>2.999592</td>
<td>1.70E-02</td>
</tr>
<tr>
<td>2</td>
<td>2.345387</td>
<td>0.337519</td>
<td>4.000000</td>
<td>0.99937</td>
<td>4.136745</td>
<td>3.901032</td>
</tr>
<tr>
<td>3</td>
<td>2.556146</td>
<td>0.000000</td>
<td>4.000000</td>
<td>0.99888</td>
<td>4.122835</td>
<td>1.068498</td>
</tr>
<tr>
<td>4</td>
<td>2.614274</td>
<td>0.333037</td>
<td>4.000000</td>
<td>0.99871</td>
<td>4.136222</td>
<td>0.871199</td>
</tr>
<tr>
<td>5</td>
<td>2.528722</td>
<td>0.441204</td>
<td>4.000000</td>
<td>0.99880</td>
<td>4.146356</td>
<td>0.344318</td>
</tr>
<tr>
<td>6</td>
<td>2.507706</td>
<td>0.526858</td>
<td>4.000000</td>
<td>0.99884</td>
<td>4.156351</td>
<td>0.264178</td>
</tr>
<tr>
<td>7</td>
<td>2.483171</td>
<td>0.525977</td>
<td>4.000000</td>
<td>0.99887</td>
<td>4.156247</td>
<td>1.46E-02</td>
</tr>
<tr>
<td>8</td>
<td>2.48249</td>
<td>0.525341</td>
<td>4.000000</td>
<td>0.99887</td>
<td>4.156166</td>
<td>1.59E-03</td>
</tr>
</tbody>
</table>
3. Model reduction-based constrained optimisation of large dissipative systems

<table>
<thead>
<tr>
<th></th>
<th>2.482629</th>
<th>0.525402</th>
<th>4.00000</th>
<th>0.99887</th>
<th>4.156174</th>
<th>1.52E-04</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.482617</td>
<td>0.525397</td>
<td>4.00000</td>
<td>0.99887</td>
<td>4.156174</td>
<td>1.30E-05</td>
</tr>
</tbody>
</table>

Figure 3.10: Profiles for the dimensionless concentration (a) and dimensionless temperature (b) at the reactor and the cooling zones, for the optimal $x_{2w}$. 
The convergence data for this scenario using Algorithm 3.2 are presented in Table 3.6 and the corresponding convergence curve is depicted in Figure 3.11.

**Table 3.6:** Convergence data for the 3dof case study using Algorithm 3.2

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$x_{2w_1}$</th>
<th>$x_{2w_2}$</th>
<th>$x_{2w_3}$</th>
<th>$f$</th>
<th>$|z^\top \nabla f|$</th>
<th>$|z^* p^*|$</th>
<th>Z-updates per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.00172</td>
<td>2.00000</td>
<td>2.00008</td>
<td>0.998615</td>
<td>1.71E-03</td>
<td>1.70E-02</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2.34539</td>
<td>0.33752</td>
<td>4.00000</td>
<td>0.998614</td>
<td>1.70E-03</td>
<td>4.31E+00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2.55477</td>
<td>0.41830</td>
<td>4.00000</td>
<td>0.998885</td>
<td>8.55E-04</td>
<td>4.91E-01</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2.51442</td>
<td>0.62524</td>
<td>4.00000</td>
<td>0.998838</td>
<td>7.43E-04</td>
<td>5.28E-01</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2.45740</td>
<td>0.56391</td>
<td>4.00000</td>
<td>0.998897</td>
<td>8.75E-04</td>
<td>1.64E-01</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2.47164</td>
<td>0.53210</td>
<td>4.00000</td>
<td>0.998881</td>
<td>1.09E-03</td>
<td>9.74E-02</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2.48014</td>
<td>0.52387</td>
<td>4.00000</td>
<td>0.998870</td>
<td>1.38E-03</td>
<td>2.80E-02</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2.48293</td>
<td>0.52551</td>
<td>4.00000</td>
<td>0.998867</td>
<td>1.58E-03</td>
<td>3.98E-03</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2.48260</td>
<td>0.52539</td>
<td>4.00000</td>
<td>0.998868</td>
<td>1.70E-03</td>
<td>3.15E-04</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>2.48262</td>
<td>0.52540</td>
<td>4.00000</td>
<td>0.998868</td>
<td>1.73E-03</td>
<td>1.84E-05</td>
<td>2</td>
</tr>
</tbody>
</table>
3. Model reduction-based constrained optimisation of large dissipative systems

3.7.4 Computational cost

The computational cost of the application of the optimisers outlined above to the case studies of Sections 3.7.2 and 3.7.3 is presented in Table 3.7. The corresponding source code is presented in Appendix and was compiled with the NAG® Fortran 95 compiler, using NAG® Fortran Library, Aspen® HSL (formerly Harwell Subroutine Library) and LAPACK using the default flags. It was executed on a desktop computer based on a dual-core Intel® Pentium® D 930 processor (3.40GHz) and 2GB of RAM (DDR2, 667MHz), running a 32-bit Linux distribution (Fedora Core 6).

For comparison purposes, note that the computational cost of applying a feasible-point reduced Hessian method to the 1-dof case study of Section 3.7.2 was 31.7 CPU seconds. Optimising using the conventional SQP method requires 43.68 CPU seconds to compute.

Table 3.7: Computational cost (CPU seconds) of the optimisation procedure

<table>
<thead>
<tr>
<th>Case study</th>
<th>Algorithm 3.1</th>
<th>Algorithm 3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.8 Systems involving Dirichlet boundary conditions

This section is concerned with handling of systems with Dirichlet boundary conditions. This kind of boundary conditions pose no problem neither conceptually, nor practically if the simulator used to model the system containing the Dirichlet BCs can be used in an input/output fashion for eigenanalysis; however this is not always the case. If the eigenspectrum corresponding to the physical system cannot be found, the decomposition strategy considered in the proposed algorithms fails. This apparent limitation can be treated using modified residuals on the boundaries, replacing the ones provided by the solver. This modification is external, and although the procedure can be characterised as more involving, the non-intrusive (black-box) nature of the algorithm is preserved.

A class of problems, for which the methods discussed are suitable, is systems modelled by PDEs. Feasible points are computed by solving a boundary value problem:

\[ \mathcal{L}u = G_D \quad \text{in } \mathcal{D} \]  
\[ \partial u = G_{\partial \mathcal{D}} \quad \text{in } \partial \mathcal{D} \]

Where \( \mathcal{D} \) is a region of a Euclidean space, \( \mathcal{L} \) is a – possibly nonlinear – differential operator acting upon \( u \) and \( \partial \) is a boundary operator, defined in the boundary \( \partial \mathcal{D} \) of \( \mathcal{D} \).

Typically, the system of (3.66) and (3.67) is solved computationally following discretisation over a computational mesh using an appropriate technique, such as Finite Differences, Finite Elements, Spectral Elements, Finite Volumes etc. (Quarteroni, 2009). After such
discretisation, solutions satisfying the aforementioned system are computed by solving an equation of the form:

$$G(u) = 0 \quad (3.68)$$

Where $G$ is a vector function which stems from (3.66) and incorporates (3.67). The value for this function is also referred to as residual, as this value expresses the difference between the solution at hand and a valid feasible solution, $u^*$, for which obviously $G(u^*) = 0$. If the number of discretisation nodes $N_n$ and $u \in \mathbb{R}^{N_n}$, then $G : \mathbb{R}^{N_n \times N_t} \rightarrow \mathbb{R}^{N_n \times N_t}$. At every node, a residual function $G_k : \mathbb{R}^{N_n \times N_t} \rightarrow \mathbb{R}^{N_t}$ results from the discretisation of $G$.

Most engineering applications involve differential operators of even order (Burnett, 1987), hence the highest order derivative of (3.66) is even, say $2m_o$, $m_o \in \mathbb{N}$. For such a problem, $m_o$ conditions need to be specified at every point belonging to the boundaries. A boundary condition involves the variables and/or its derivatives (of order 1 to $m_o - 1$).

Boundary condition for boundary valued problems can be classified as:

- Dirichlet (essential)
- Neumann (natural)
- Robin (mixed)

depending on the terms involved in the equation for the BCs. Dirichlet BCs involve the dependent variable ($u$) and/or its derivatives of order 1 to $m_o - 1$. Neumann BCs only involve derivatives of order $m_o$ to $2m_o - 1$ whereas Robin BCs involve derivatives of order 1 to $2m_o - 1$ and potentially the $u$ as well. The latter type of BCs is typically treated as Neumann BCs.

The model for tubular reactors, which was used in previous Section, only includes Neumann and not Dirichlet BCs. However many engineering problems do. Dirichlet boundary conditions are equations of the form

$$u = \alpha_{\partial D}, \text{ at } \partial D \quad (3.69)$$

Where $\alpha_{\partial D}$ is a constant.

Sometimes, the essential BCs are implemented in the computational model by replacing the corresponding elements of $G(u)$ of Equation (3.66), by the expression $u - \alpha_{\partial D}$, resulting
from (3.69). The resulting system has the same solution as the original one, but cannot be used for eigenanalysis. This approach poses a problem when used in conjunction with the proposed algorithms, as the basis for the model reduction, is the identification of a basis for the subspace spanning the eigenspace belonging to specific eigenvalues. Since the eigenspectrum of the system corresponding to the computational model is altered, such a basis cannot be computed in the same way.

The eigenspectrum corresponds to the system dynamics. Especially for the case of linear systems, the time evolution of the dependent variables can be expressed as a function of the eigenvectors (Lynch, 2009). The eigenvectors span the solution space and give insights on the physical characteristics of the system. Thus, the eigenspectrum is altered by replacing the equations of model of the system with others which have the same solution but do not exhibit the same dynamics. In specific, a model which implements Equation (3.69) would always have 1 as an eigenvalue. Even if the simulator is a true black-box and there is no information regarding how Dirichlet BCs are handled, if one or more eigenvalues of the value of exactly 1 are obtained by performing matrix-free eigenanalysis tasks, it is an indication that the Dirichlet BCs are implemented in a way which prevents eigenanalysis.

To illustrate the considerations presented, the model for the tubular reactor is used, where an exothermic, first-order irreversible reaction takes place: $A \rightarrow B$ (Jensen and Ray, 1982), which used in Section 3.7.

The boundary conditions that this case study has are natural (von Neumann). Here a Dirichlet condition is enforced on the left boundary instead of the Neumann one used previously $\frac{\partial x_1}{\partial y} - P e x_1 = 0$. This corresponds to setting the temperature in the input of the reactor to a specific value, which is a typical case in practice. The new BCs are:

$$\begin{align*}
\frac{\partial x_1}{\partial y} - P e x_1 &= 0, \quad x_1 = 0.216146 \quad \text{at} \quad y = 0 \\
\frac{\partial x_1}{\partial y} &= 0, \quad \frac{\partial x_2}{\partial y} = 0 \quad \text{at} \quad y = 1
\end{align*}$$

(3.70)

For $Da = 0.1220$, the leading eigenvalues obtained by replacing the function at the boundary with the essential condition, are: $1.0000, -0.2405 \pm 1.5714i, 4.8407 \pm 2.8568i, -12.0015 \pm 3.3940i, -22.7275 \pm 3.6588i, -37.4325 \pm 3.7892i$, whereas if the boundary
condition is passed on properly, the eigenvalues are: 

\[-0.2411\pm 1.5590i, \quad -4.8059\pm 2.8379i, \quad -11.9110\pm 3.3902i, \quad -22.5474\pm 3.6880i, \quad -37.1278\pm 3.8928i.\]

Although the eigenvalues differ on average by less than $10^{-3}$, i.e. the average norm of the difference over the norm of the eigenvalue is less than 0.1%, the corresponding eigenvectors differ by 29.9%. Hence the directions provided by the eigenvectors in the first case do not span the null space with adequate accuracy.

To address the issue of identifying a basis for the dominant subspace of the system in the presence of essential BCs, one can identify the elements of the function which correspond to the Dirichlet boundaries, and replace them externally with finite difference approximations. This procedure does not violate the black-box nature of the proposed algorithm, since the equations of the model are known even in the case of black box solvers. Other information that need be used are the solution values at specific points and the spatial coordinates of each node.

Passing the Dirichlet BC in the “simplistic way” (Equation (3.69)) and using the model for optimisation, convergence is achieved with a conventional reduced Hessian method to $Da = 0.12236$ in 8 iterations, whereas the proposed optimisation method fails to converge. If one would choose to approximate the Dirichlet BCs externally as described previously, the proposed method would converge to $Da = 0.12177$ in 14 iterations.

### 3.9 Summary

In this chapter, a novel optimisation algorithm for large-scale dissipative nonlinear systems has been presented. It employs model reduction technology based on the dissipativity of the underlying system, manifesting as a separation of eigenvalues in the spectrum of the linearised system. In specific, a small number of eigenvalues seems to form a cluster with algebraically large real part, which can be discerned from the bulk of eigenvalues. A spectral gap separates the two groups of modes. The subspace belonging to the rightmost cluster of eigenvalues can be considered as dominant, since heuristically it can be used to sufficiently the whole kernel space. The main feature of the new method is a two-step
projection scheme, first onto the dominant subspace of the system and second onto the subspace of the decision variables. In the proposed framework only low-dimensional Jacobians and Hessians are involved, which are projections of the original matrices onto appropriate subspaces, efficiently numerically computed with directional perturbations. The proposed scheme is matrix-free and has low requirements in memory requirements and computing power than conventional deterministic optimisation methods. Illustrative case studies have been presented to show the behaviour of the proposed optimisation scheme, which are based on the tubular reactor. Furthermore, techniques for enhancing its efficiency have been presented. Issues that might arise in systems which involve Dirichlet boundary conditions and therefore may pose difficulty in using for eigenanalysis have also been dealt with.
4. Large-scale nonlinear programming of dissipative systems with nonlinear inequality constraints

4.1 Introduction

In Chapter 3, a new optimisation technique has been presented for large scale dissipative systems. It exploits the dissipative nature of the underlying phenomena for model order reduction (MOR). This property is expressed as separation of eigenvalues in the spectrum of the linearised system and therefore separation of modes to slow and fast ones. This separation has been used in various ways within MOR context, leading to different formulations (see Chapter 2). In this Chapter, extensions to the proposed optimisation framework are provided, in order to enable it to deal with nonlinear programming (NLP) problems which include nonlinear inequality constraints. Such constraints may express manufacturing, economic, environmental or safety limitations, which the process needs to meet. Caution is advisable when treating such problems, as global parameterization may lead to instabilities (Gurwitz and Overton, 1989). In this Chapter, two approaches are examined and combined with the algorithms presented in Chapter 3: the partial reduction approach (Schulz, 1998), which only considers the equality constraints in order to reduce the QP subproblem and the constraint aggregation approach (Itle et al., 2004), which replaces all inequalities with an overestimating function and adopts a barrier approach. In
both formulations proposed, MOR techniques are applied to equality constraints only, for reasons which will be outlined in Section 4.2.

The optimisation framework presented in Chapter 3 features a two-step projection scheme, firstly onto the adaptively identified dominant subspace of the system, and secondly onto the subspace of decision variables. Provided that the dimension of the dominant subspace is relatively small in comparison to the dimension of the full problem and that a basis for it can be computed at a reasonable cost, the 2-step projection scheme leads to savings in both computational expense and memory requirements. This is due to the fact that the calculation and inversion of the full Jacobian at every iteration of the optimisation algorithm is evaded. Instead, an efficiently calculated projection onto the dominant subspace of the system is used. In the same spirit, the Lagrange multipliers are also projections onto the dominant subspace. It is evident that this projection is not as accurate as the full Jacobian. The accuracy by which one can reproduce the full Jacobian by its low-order counterpart depends on the “quality” of the dominant subspace, i.e. how good an approximation to the kernel space this subspace constitutes. It is a well-known fact that reduced Hessian methods require the existence of accurate Jacobian (Biegler et al., 2003). However, in large-scale systems, usually the Jacobian used for simulation purposes results from an analytical differentiation procedure of the discretised PDEs ignoring cross-terms. Thus, in the conventional reduced Hessian method, one can either compute an exact Jacobian at an extra computational cost or use the one provided, which is also an approximation, much like the projected one.

Matrix-free methods are employed for the computations of projected Jacobians, Hessians and bases for the dominant subspaces, which lead to efficient computations and good scaling up with problem size. Even if the clustering of eigenvalues of the eigenspectrum (and consequently the spectral gap) is not very clear by inspection, the existence of a dominant subspace may be verified numerically in various points of the parameter space. The basis for this subspace is updated at every iteration of the optimisation algorithm, rendering its computation adaptive. The computation of the reduced Hessian is performed with directional perturbations rather than estimated using a Broyden – Fletcher – Goldfarb – Shanno (BFGS) update scheme. This leads to faster convergence if the reduced Hessian is positive definite, but the procedure might fail otherwise. In such cases, corrections to the Hessian may be applied (cf. Fletcher, 1981).
The proposed optimisation framework is appropriate to link with existing simulators. It requires minimal information from the simulator, as it treats is in an input/output fashion. This characteristic renders it appropriate for use in conjunction with commercial software or in-house developed legacy codes, which do not perform optimisation tasks. This is an important feature, since this simulator may represent many man-years of work and be accurate, so that extending its use for design purposes would have obvious advantages.

The proposed method is implemented as a feasible point algorithm, calling the existing simulator to compute a feasible solution after every optimisation step and thus belonging to the Nested Analysis and Design class of problems. Convergence of the nested simulation problem could potentially be expensive, so an infeasible path implementation may be preferred. This may render the algorithm more intrusive to the simulator. Reduced Hessian methods can either be implemented following a direct tailored approach, or an adjoint tailored approach (Biegler and Wachter, 2003). In the former, the existing simulator is used for the direct computation of reduced gradients and search directions, whereas in the latter those quantities are computed using both the system Jacobian and its transpose (Biegler and Wachter, 2003).

The rest of the chapter is organized as follows: In Section 4.2 the nonlinear programming problem including nonlinear inequalities is introduced. This is a generalization of the problem considered in the previous chapter. Next, two approaches will be presented in Sections 4.3 and 4.4 for solving this problem by extending the methodology presented in the previous Chapter. The extensions presented are applied for a case study based on the tubular reactor, which is presented in Section 4.5.

### 4.2 Large-scale inequality constrained nonlinear programming

The general formulation for NLP problems has been defined in the previous Chapter (Equation (3.7)):
4. Large-scale NLP of dissipative systems with nonlinear inequality constraints

\[
\min f(x) \\
\text{s.t. } G(x) = 0 \\
h(x) \leq 0 \\
x^l \leq x \leq x^u
\]

(4.1)

where \( f: \mathbb{R}^{N_{dof}} \to \mathbb{R} \) is the objective function, \( G: \mathbb{R}^{N_{dof}} \to \mathbb{R}^N \) is the vector function of the equality constraints, \( h: \mathbb{R}^{N_{dof}} \to \mathbb{R}^{N_u} \) comprises the inequality constraints, \( x \in \mathbb{R}^{N_{dof}} \) is the vector of the dependent \((u \in \mathbb{R}^N)\) and independent variables \((z \in \mathbb{R}^{dof})\): \( x^T = [u^T \ z^T]^T \), with \( N, N_u, dof \in \mathbb{N} \).

Not all NLP problems that arise in engineering practice contain nonlinear inequality constraints. These may express manufacturing, economic, environmental or safety limitations. In many problems, the limiting conditions expressed via inequalities are reduced to simple bounds for variables. These cases do not pose a problem, as optimisers which are suitable only for equality constrained problems are able to handle bounds. Some classes of inequality constraints (such as ratios of variables) can be seamlessly reduced to simple bounds on existing variables (Farias et al., 2007) or alternatively in some classes of problems (such as the parameter estimation problems solved with the errors in variables method) the inequality constraints can be directly incorporated in the model parameters (Tjoa and Biegler, 1991).

SQP schemes handle inequalities following EQP or IQP approaches (see section 3.1). However they are not appropriate for large-scale systems (Nocedal and Wright, 1999). Some methodological approaches for handling inequality constraints were reviewed in Section 2.3. Applications with such constraints include among others shape optimisation (Diehl et al., 2002), optimisation of chemical vapour deposition processes (Itle et al., 2004), thin film growth processes in general (Christofides et al., 2009), particulate systems (Shi et al., 2006), polymerization reactors (Zavala and Biegler, 2009a), catalytic distillation columns (Jäger et al., 2007) and so forth. An important class of problems which inequality constraints naturally arise is dynamic optimisation and nonlinear model predictive control (Schäfer et al., 2007).

As mentioned in the introduction of this chapter, two approaches are examined and combined with the optimisation framework of Chapter 3, in order to enable it to handle nonlinear inequalities. These strategies can be applied to the static optimisation algorithm employing dynamic simulators of Luna-Ortiz and Theodoropoulos (2005) and the dynamic
optimisation algorithm of Theodoropoulos and Luna-Ortiz (2006) to enable them to handle such constraints. Hence, here a framework extending the class of model reduction-based optimisation algorithms of Theodoropoulos and co-workers (Luna-Ortiz and Theodoropoulos, 2005; Theodoropoulos and Luna-Ortiz, 2006; Bonis and Theodoropoulos, 2008; Theodoropoulos, 2011) is provided. This synopsizes the novelty of the work presented in this Chapter.

The straightforward approach of introducing slack variables and treating inequalities as equalities is not always the best tactic. The strategies that were chosen are: partial reduction only with respect to the equality constraints and utilizing conservative barrier functions. More information will be given in the next two Sections. If the active inequality constraints were treated as equalities and considered for model reduction, two undesirable side effects would be possible:

- The dimension of the basis would vary during run-time, depending on the number of active inequalities. In such a case it is difficult to provide a good initial guess for the basis for the dominant subspace. Hence the numerical efficiency would be jeopardized.
- The augmentation of the equality constraints with inequalities may have perturbed the eigenspectrum and ruin the separation of scales due to the addition of non-dissipative modes coming from the active inequalities.

4.3. The notion of Partial Reduction in tackling with inequality constraints

Partially Reduced Sequential Quadratic Programming (PRSQP) has been introduced by Schulz (1996) in order to extend the reduced Hessian method for cases where the optimisation problem involves nonlinear inequality constraints and/or additional equality constraints which are not considered within the separability framework. Thus, the optimisation defined by Equation (4.1) which involves inequality constraints is a subset of the problems PRSQP can deal with.
The main idea of PRSQP follows the approach of reduced space SQP, but only the kernel subspace of the equality constraints of the NLP is used for the reduction. Thus, the additional equality and inequality constraints are not eliminated in the quadratic subproblem. Rather, linearised projections of the corresponding expressions onto the subspace considered are passed on to the QP subproblem as extra equalities and inequalities correspondingly. Furthermore, the Lagrange function includes terms for the additional equality as well as the inequality constraints and additional Lagrange multipliers which correspond to these terms.

This approach has been followed in the context of the RPRH optimisation delineated in the previous chapter. The reduction is onto the dominant space of $G$, as in the standard algorithm. The main features of the strategy, the two-step projection scheme as well as the computation of the bases for the two subspaces involved in the projection, remain unchanged. The difference is traced to the definition of the Lagrange function which also affects the computation of the reduced Hessian. It can be shown, that the Lagrange function can be calculated as:

$$L(x, \lambda) = f(x) + \phi^T \dot{Z}^T G(x) + \lambda_{ineq}^T h(x)$$  \hspace{1cm} (4.2)

Where the Lagrange multipliers corresponding to the inequality constraints, are computed directly from the solution of the QP subproblem. The Lagrange multipliers corresponding to the equality constraints are projected onto the dominant subspace as in the standard algorithm, but are estimated using:

$$-H \phi = \dot{Z}^T (y^T \nabla f + \lambda_{ineq}^T h)$$  \hspace{1cm} (4.3)

Furthermore, the QP subproblem, with the addition of the projections of the linearised inequality constraints, and ignoring the range space components as mentioned in the introductory section, is:

$$\min_{p_z} \left( Z^T \nabla f_p \right)^T p_z + 1/2 p_z^T \theta_p p_z$$

s.t.  $\nabla h(x) Z^T p_z + h(x) \leq 0$

$$\left(x^i - x\right) \leq Z^T p_z \leq \left(x^u - x\right)$$  \hspace{1cm} (4.4)

The methodology analyzed can be condensed in an algorithm, which is presented in Algorithm 4.1.
4. Large-scale NLP of dissipative systems with nonlinear inequality constraints

**Table 4.1: Algorithm 4.1:** the proposed optimisation scheme with inequalities being handled by partial reduction

1. Choose initial guesses for \( x_0 \) and \( B_0 \) (initial guess)

2. Calculate the steady state with the input/output simulator and compute \( x \) and \( f \)

3. Calculate a basis for the dominant subspace, \( \hat{Z} \)

4. Compute the reduced Jacobian of the equality constraints, \( H \) from \( H = \hat{Z}^T J \hat{Z} \) with numerical perturbations

5. Calculate a basis for the subspace of the independent variables, \( Z^* \):

\[
Z^* = \hat{Z}_{\text{red}} Z_r = \begin{bmatrix} -\hat{Z} H^{-1} \nabla^T \phi^* \end{bmatrix}_I
\]

6. Compute the reduced Hessian, \( \hat{B}_r = Z^T B Z^* \) by numerically perturbing the Lagrange function in the direction \( Z^* \) and using the reduced Lagrange multipliers, \( \phi \)

7. Find the search direction, \( p_z \), and the Lagrange multipliers corresponding to the inequality constraints, \( \lambda_{\text{inv}} \) by solving the QP subproblem:

\[
\begin{align*}
\min_{p_z} & \left(Z^T \nabla f_z\right)^T p_z + 1/2 p_z^T B_r p_z \\
\text{s.t.} & \ (x^l - x) \leq Z^* p_z \leq (x^u - x), \\
& \nabla h(x) Z^* p_z + h(x) \geq 0
\end{align*}
\]

8. Update the solution: \( x = x + Z^* p_z \)

9. Update \( \hat{Z} \) and calculate the corresponding \( H \) as per step 4.

10. Calculate the estimates of the reduced Lagrange multipliers corresponding to the equality constraints, \( \phi \): \( H^T \phi = -\hat{Z}^T y^T \nabla f \)

11. Assume that convergence has been achieved if: \( \frac{\|x_t - x_{t-1}\|}{\|x_t\| + 1} < \alpha \). If not, go
4.4. Handling inequalities with constraint aggregation functions

The second approach followed in order to enable the proposed optimisation method to handle inequality constraints is based on the use of a penalty function. In specific, the Kresselmeier – Steinhauser (KS) function is used to aggregate all inequality constraints and replace them by an overestimator: $KS(\mathbf{x}, \rho) \geq \max_{i} (h_i(\mathbf{x}))$, $\rho \in \mathbb{R}_+$ (Raspanti et al., 2000; Itle et al., 2004) This approach does not presuppose that the inequality constraints are continuously differentiable. The KS function has two equivalent definitions:

$$KS(h_j) = \frac{1}{\rho} \ln \left[ \sum_{j=1}^{N_{in}} \exp \left( \rho h_j \right) \right]$$  \hspace{1cm} (4.5)

and

$$KS(h_j) = M + \frac{1}{\rho} \ln \left[ \sum_{j=1}^{N_{in}} \exp \left( \rho (h_j - M) \right) \right], \hspace{0.5cm} M \approx \max(h_j)$$  \hspace{1cm} (4.6)

Where $j = 1, \ldots, N_{in}$ $N_{in}$ is the number of nonlinear inequality constraints, $\rho$ and $M$ are scalar parameters. Since $\lim_{\rho \to \infty} KS(h_j) = \max(h_j)$, large values of $\rho$ are preferred. The use of Equation (4.6) is preferred as it leads to more numerically robust computations in cases where the exponential term has a very high value. For a more complete recitation of the properties of the KS function as well as guidelines for the choice of the parameters $\rho$ and $M$, see (Raspanti et al., 2000).

Numerical experience with this scheme has shown that it is advantageous to perform scaling of the variables when the KS approach is followed. Thus, the scaled variables $\hat{x}$ are defined as:
The KS function can then be evaluated using Equation (4.6). This value can be incorporated to a new merit function, $f_{\mu}$.

$$f_{\mu}(x) = f(x) + KS(h_i)$$  

The merit function of Equation (4.8) is used as objective in a modified optimisation problem that only includes the equality constraints and the bounds for the variables. This equality-constrained optimisation problem can be solved with the standard optimisation method outlined in Chapter 3. The resulting algorithm enabling the handling of inequality constraints using a KS function is delineated in Algorithm 4.2

**Table 4.2: Algorithm 4.2:** the proposed optimisation scheme with inequalities being handled with a KS function

1. Define $x_0$ and $B_0$ (initial guess)

2. Calculate the steady state with the input/output simulator and compute the dependent variables $x$ and the objective function $f$.

3. Define the scaled dependent variables: $\bar{x}_i = \frac{x_i}{x_i^U}$, $i = 1, ..., N$ and evaluate the merit function incorporating the aggregated constraints $f_{KS} = f + \frac{1}{\rho} \ln \left[ \sum_{i=1}^{N} e^{h_i(x)} \right]$.

4. Calculate a basis for the dominant subspace, $\hat{Z}$

5. Compute the reduced Jacobian of the equality constraints, $H$ from $H = \hat{Z}^T J \hat{Z}$ with numerical perturbations

6. Calculate a basis for the subspace of the independent variables, $Z^*$:

$$Z^* = \hat{Z}_{\text{ext}} Z_e = \left[ -\hat{Z} H^{-1} \hat{Z}^T \nabla \mu^T G^T \right]$$
7. Compute the reduced Hessian, $\tilde{\mathbf{B}} = Z^T \mathbf{B} Z^T$, by numerically perturbing the Lagrange function in the direction $Z^*$ and using the reduced Lagrange multipliers, $\varphi$.

8. Find the search direction, $\mathbf{p}_z$, by solving the QP subproblem:

$$
\min_{\mathbf{p}_z} \left( Z^T \nabla f(\mathbf{x}) \right)^T \mathbf{p}_z + 1/2 \mathbf{p}_z^T \mathbf{B}_z \mathbf{p}_z, \\
\text{s.t.} \ (\mathbf{x}^i - \mathbf{x}) \leq Z^* \mathbf{p}_z \leq (\mathbf{x}^u - \mathbf{x})
$$

9. Update the solution: $\mathbf{x} = \mathbf{x} + Z^* \mathbf{p}_z$

10. Update $\tilde{Z}$ and calculate the corresponding $H$ as per step 4.

11. Calculate the estimates of the reduced Lagrange multipliers for the next iteration, $\varphi$: $H^T \varphi = -\tilde{Z}^T \nabla f$

12. Assume that convergence has been achieved if: $\frac{\|\mathbf{x}_k - \mathbf{x}_{k-1}\|}{\|\mathbf{x}_k\| + 1} < \varepsilon_1$. If not, go to step 2.

4.5. Case study

To illustrate the behaviour of the proposed optimisation algorithms, they are applied for optimisation of tubular reactors, which are known to exhibit a rich parametric behaviour. The base-case is the same as in the previous Chapter, however the equations will be repeated for completeness. A reactor, where an irreversible, first order, exothermic reaction takes place: $A \rightarrow B$ is considered. A schematic of the reactor with three heat exchangers on its jacked is shown in Figure 4.1. The model of the reactor consists of 2 PDEs (Jensen and Ray, 1982)
4. Large-scale NLP of dissipative systems with nonlinear inequality constraints

\[ \frac{\partial x_1}{\partial t} = \frac{1}{Pe_1} \frac{\partial^2 x_1}{\partial y^2} - Da(1-x_1) \exp \left( \frac{x_2}{1+x_2/y} \right) + x_1 \]

\[ \frac{\partial x_2}{\partial t} = \frac{1}{LePe_2} \frac{\partial^2 x_2}{\partial y^2} - \frac{1}{Le} \frac{\partial x_2}{\partial y} - \frac{\beta}{Le} x_2 + \frac{C}{Le} Da(1-x_1) \exp \left( \frac{x_2}{1+x_2/y} \right) + \beta x_{2w} \]

Where \( x_1 \) is the dimensionless concentration of the product, \( x_2 \) is the dimensionless temperature of the reactor, \( Da \) is the Damköhler number, \( Le \) is the Lewis number, \( Pe_1 \) is the Pecklet number for mass transport and \( Pe_2 \) for heat transport, \( \beta \) a dimensionless heat transfer coefficient, \( C \) is the dimensionless adiabatic temperature rise, \( y \) the dimensionless activation energy, \( y \) the dimensionless longitudinal coordinate and \( x_{2w} \) the dimensionless adiabatic wall temperature which varies spatially:

\[ x_{2w}(y) = \sum_{j=1}^{3} (H(y-y_{j-1}) - H(y)-y_j) \]

Where \( z_j, j = 1, 2, 3 \) is the independent variable. \( y_i = i / 3, i = 0, 1, ..., 3 \), \( x_{2w}, j = 1, 2, ..., 3 \), and \( H(\cdot) \) is the Heaviside step function:

\[ H(y) = \begin{cases} 0, & y < 0 \\ 1, & y \geq 0 \end{cases} \]

\[ \text{Figure 4.1: Schematic representation of a tubular reactor with three cooling zones} \]

At steady-state, the set of Equations (4.9) becomes:

\[ \frac{1}{Pe_1} \frac{\partial^2 x_1}{\partial y^2} - \frac{\partial x_1}{\partial y} + Da(1-x_1) \exp \left( \frac{x_2}{1+x_2/y} \right) = G_1 = 0 \]

\[ \frac{1}{LePe_2} \frac{\partial^2 x_2}{\partial y^2} - \frac{1}{Le} \frac{\partial x_2}{\partial y} - \frac{\beta}{Le} x_2 + \frac{C}{Le} Da(1-x_1) \exp \left( \frac{x_2}{1+x_2/y} \right) + \beta x_{2w} = G_2 = 0 \]

The set of equations (4.12) is discretised with the central Finite Differences method on a mesh of 250 nodes. This discretisation results in a 500 dependent variables.
As in the case study of the Section 3.7, the boundary conditions for the above set of nonlinear equations are:

\[
\frac{\partial x_1}{\partial y} - Pe_1 x_1 = 0, \quad \frac{\partial x_2}{\partial y} - Pe_2 x_2 = 0 \quad \text{at} \quad y = 0
\]

\[
\frac{\partial x_1}{\partial y} = 0, \quad \frac{\partial x_2}{\partial y} = 0 \quad \text{at} \quad y = 1
\]

(4.13)

The values of the parameters chosen are \( Le = 1.0, Pe_1 = Pe_2 = 5.0, \gamma = 20.0, \beta = 1.50, C = 12.0. \)

The objective of the optimisation problem is the maximization of the concentration at the exit from the reactor, with respect to the temperatures of the 3 cooling zones:

\[
\max_{x} x_1 \big|_{\text{exit}}
\]

\[
\text{s.t. } G(x) = 0
\]

\[
0 \leq x_1 \leq 1
\]

\[
0 \leq x_2 \leq 8
\]

\[
0 \leq z_j \leq 4, \quad j \in \{1, 2, 3\}
\]

(4.14)

The optimisation problem described so far does not include nonlinear inequality constraints and can be solved using the standard version of the optimisation algorithm presented in the previous Chapter; in that case, convergence can be achieved in 10 iterations.

In order to be able to test the behaviour of the proposed schemes when handling inequality constraints, the NLP (4.14) is augmented by nonlinear inequality constraints. In specific, an area close to the middle of the reactor is considered, where nonlinear inequalities have to hold:

\[
h(x) = \left( x_2 \big|_{k} + x_2 \big|_{k+1} \right)^2 \leq 160, \quad k = \{125, 127, 129\}
\]

(4.15)

Let it be noted that the inequality constraints added to the original NLP were such that the optimal solution for the base scenario did not satisfy. The purpose if this choice is to illustrate the behaviour of the Algorithms 4.1 and 4.2 in cases with active inequalities active. The results from the optimisation in terms of values of values of the independent variables and objective function as well as number of iterations are presented in Table 4.3.
Table 4.3: Results obtained with different strategies

<table>
<thead>
<tr>
<th>Solution strategy</th>
<th>$x_{2w_1}$</th>
<th>$x_{2w_2}$</th>
<th>$x_{2w_3}$</th>
<th>$f$</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>No inequalities (base scenario)</td>
<td>2.487</td>
<td>0.5253</td>
<td>4.000</td>
<td>0.9989</td>
<td>10</td>
</tr>
<tr>
<td>Algorithm 4.1 (PRSQP)</td>
<td>2.6173</td>
<td>0.0000</td>
<td>3.1237</td>
<td>0.9984</td>
<td>7</td>
</tr>
<tr>
<td>Algorithm 4.2 (KS)</td>
<td>1.9021</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.9940</td>
<td>8</td>
</tr>
</tbody>
</table>

The convergence data for the optimisation procedures with Algorithms 4.1 and 4.2 are presented in Tables 4.4 and 4.5 correspondingly. Figures 4.2 and 4.3 constitute the convergence curves for Algorithms 4.1 and 4.2 respectively and Figure 4.4 presents the solution profiles for the optimal values of the independent variables found. In the solution found using Algorithm 4.1 only the first nonlinear inequality constraint is active, whereas using Algorithm 4.2, the first two constraints are active.

Table 4.4: Convergence data using Algorithm 4.1

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$x_{2w_1}$</th>
<th>$x_{2w_2}$</th>
<th>$x_{2w_3}$</th>
<th>$f$</th>
<th>$|z^*\nabla f|$</th>
<th>$|z^*p_i|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0017</td>
<td>1.9986</td>
<td>1.9992</td>
<td>0.9986</td>
<td>0.0017</td>
<td>0.0199</td>
</tr>
<tr>
<td>2</td>
<td>2.1555</td>
<td>0.0000</td>
<td>3.2638</td>
<td>0.9990</td>
<td>0.0020</td>
<td>4.0259</td>
</tr>
<tr>
<td>3</td>
<td>2.6659</td>
<td>0.0000</td>
<td>3.1102</td>
<td>0.9986</td>
<td>0.0014</td>
<td>0.9678</td>
</tr>
<tr>
<td>4</td>
<td>2.6068</td>
<td>0.0000</td>
<td>3.1327</td>
<td>0.9984</td>
<td>0.0015</td>
<td>0.1493</td>
</tr>
<tr>
<td>5</td>
<td>2.6164</td>
<td>0.0000</td>
<td>3.1307</td>
<td>0.9984</td>
<td>0.0015</td>
<td>0.0177</td>
</tr>
</tbody>
</table>
4. Large-scale NLP of dissipative systems with nonlinear inequality constraints

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$x_{2w_1}$</th>
<th>$x_{2w_2}$</th>
<th>$x_{2w_3}$</th>
<th>$f_{KS}$</th>
<th>$f$</th>
<th>$|z^T \nabla f|$</th>
<th>$|z^T p_2|$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.9445</td>
<td>1.8685</td>
<td>1.9720</td>
<td>0.3932</td>
<td>0.9985</td>
<td>0.1429</td>
<td>0.7828</td>
</tr>
<tr>
<td>2</td>
<td>1.4997</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0832</td>
<td>0.9973</td>
<td>0.1245</td>
<td>10.5719</td>
</tr>
<tr>
<td>3</td>
<td>2.2007</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0981</td>
<td>0.9955</td>
<td>0.1306</td>
<td>1.0503</td>
</tr>
<tr>
<td>4</td>
<td>2.0340</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0918</td>
<td>0.9946</td>
<td>0.1303</td>
<td>1.8852</td>
</tr>
<tr>
<td>5</td>
<td>1.9706</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0930</td>
<td>0.9944</td>
<td>0.1306</td>
<td>0.5326</td>
</tr>
<tr>
<td>6</td>
<td>1.9334</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0925</td>
<td>0.9942</td>
<td>0.1309</td>
<td>0.4309</td>
</tr>
<tr>
<td>7</td>
<td>1.9111</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0924</td>
<td>0.9941</td>
<td>0.1322</td>
<td>0.4762</td>
</tr>
<tr>
<td>8</td>
<td>1.9021</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0925</td>
<td>0.9940</td>
<td>0.1316</td>
<td>0.1507</td>
</tr>
</tbody>
</table>

Table 4.5: Convergence data using Algorithm 4.2

Although the two sets of independent variables computed from the two algorithms seem very different (i.e. the norm of their difference is large), the corresponding objective values are very similar. Obviously Algorithm 4.2 does not minimise the actual objective function, but a merit function which also includes the value of the constraint aggregation function. Thus, the effect of the penalty term is expected in the results. This effect could be of high or low impact on the results, depending on the values for $M$ and $\rho$ chosen. Those values could be fixed, or could vary depending on the iteration number and the values of the slack variables (i.e. the margin by which the inequality constraints are satisfied).
The two algorithms produce solutions that are noticeably different as by depicted by Figure 4.4. Nevertheless, both solutions are feasible: they satisfy the equality constrains and barely violate one or two constraints (by $O(10^{-7})$); this is not considered a violation, rather the respective constrains are considered active. Intuitively this behaviour is expected: the algorithms are leading the system towards the direction of active inequalities so that the minimisation of the objective function is achieved. The different profiles are a result of the different values of the decision variables found; nevertheless both sets of values result in similar values of the objective function (0.9989 and 0.9984). Even greater differences would be justified, since Algorithm 4.2 follows a barrier approach, which alters the objective function, hence its gradient and consequently the optimisation path.

The computational cost of the application of the optimisers outlined above to the case study presented above was 9.588 CPU seconds using Algorithm 4.1 and 12.794 CPU seconds using Algorithm 4.2. The corresponding source code is presented in Appendix and was compiled with the NAG® Fortran 95 compiler, using NAG® Fortran Library, Aspen® HSL (formerly Harwell Subroutine Library) and LAPACK using the default flags. It was executed on a desktop computer based on a dual-core Intel® Pentium™ D 930 processor (3.40GHz) and 2GB of RAM (DDR2, 667MHz), running a 32-bit Linux distribution (Fedora Core 6).

![Figure 4.2: The convergence curve using Algorithm 4.1 (PRSQP).](image)
4. Large-scale NLP of dissipative systems with nonlinear inequality constraints

4.6. Summary

The model reduction-based deterministic optimisation method for large-scale dissipative PDE-constrained optimisation which was presented in Chapter 3 has been extended for nonlinear programming problems which involve inequality constraints. The new algorithms presented, like the standard one, feature is a 2-step projection scheme. The first projection is onto the (considered to be) dominant subspace of the system at hand, augmented by the subspace of the independent variables, and the second is onto the tangent space of the equality constraints. The underlying hypothesis is that the solutions of dissipative systems can be reproduced using a relatively small number of eigenvectors (representing modes), which are thus treated as basis functions. Two approaches have been followed to deal with inequality constraints: partial reduction of the nonlinear constraints of the system, and a penalty function-based approach. In the first approach, only the equality constraints are used to produce a tangent space, on which the quadratic subproblems are projected. The inequalities are linearised and passed to the QP.

Figure 4.3: The convergence curve using Algorithm 4.2 (KS).
Figure 4.4: Dimensionless concentration (a) and temperature (b) profiles at the optimum using Algorithms 4.1 and 4.2 plotted versus the base scenario without inequalities.
On the other hand, the formulation in the penalty function-based approach is closer to the original algorithm. The constraints are aggregated using a KS (overestimation) function and then included in a new objective (merit) function, that is used in the context of optimisation using the standard algorithm. The behaviour of the two approaches is illustrated by applying them for the optimisation of a tubular reactor.
5. “Equation-free” linear Model Predictive Control of nonlinear Distributed Parameter Systems

5.1 Introduction

In previous chapters, large-scale optimisation methods have been presented for nonlinear dissipative systems. Model order reduction (MOR) plays a key role in the proposed methods. In this chapter, the same MOR notions are applied for the construction of efficient control algorithm. The class of systems examined is the same as in previous chapters: large-scale systems, which typically result from the discretisation of Distributed Parameter Systems (DPS) that exhibit dissipation. The control of such systems is challenging. Conventional control methods may prove inefficient or inapplicable due to their large size and the distributed nature of actuation (Christofides, 2001).

It is fortunate that the dynamics of such systems are usually dominated by a finite – and usually small – number of modes. This observation has been exploited in many ways, both following systematic and heuristic paradigms. Systematic attempts for the control of DPS include the utilization of model reduction technology in order to obtain a low-dimensional model, which can further be used for linear or nonlinear automatic control. Model reduction methodologies which have successfully been applied in conjunction with control techniques include Galerkin’s method (Shi et al., 2006), Proper Orthogonal Decomposition (or Karhunen – Loeve expansion) (Shvartsman and Kevrekidis, 1998; Shvartsman et al., 2000; Armaou and Christofides, 2002) approximate inertial manifold methods (Christofides
and Daoutidis, 1997; Ito and Kunisch, 2008), reduced Lagrange basis methods (Ito and Ravindran, 1998), modal decomposition (Dubljevic and Christofides, 2006), and the equation-free methods (Armaou et al., 2004b; Armaou et al., 2005; Siettos et al., 2006).

Various control schemes appropriate for DPS have been proposed (Dufour et al., 2003). Since these techniques usually include optimisation procedures, the design of new tailor-made algorithms has been induced (Zavala et al., 2008; Biegler, 2009). The identification of linear or nonlinear models approximating the behaviour of the underlying system at a certain region of operation has also been proposed (Panos et al., 2010). Among the metamodels used for system identification, special mention needs be given to stochastic and hybrid models (Aggelogiannaki and Sarimveis, 2008).

The simplest tactic is to replace the nonlinear model with a linear one. Nevertheless, this strategy does not always produce satisfying results, as the surrogate model cannot generally express the nonlinear behaviour of the underlying system over sizeable regions (Henson, 1998). Some advanced control algorithms do allow the direct use of nonlinear models. Such an algorithm is nonlinear MPC. The increased accuracy is an obvious advantage, but it comes at a cost, as the computational cost could be excessive for large systems and issues of uniqueness of the optimum control action arise. Employing multiple linear models within linear MPC has been proposed as a compromise between the two aforementioned approaches (Yu et al., 1992; Mazinan and Sadati, 2009). Issues that might arise in such formulation concern the determination of the bank of models and strategies for switching between them. Finally, another approach is to perform successive linearisation of the nonlinear model on a nominal trajectory and use the sequence of linear models produced in the context of predictive control. The accuracy of the resulting algorithms is comparable to NMPC, but the optimisation subproblems produced are quadratic. However, the need for a nominal trajectory may be considered as a limitation.

In this Chapter, a novel model reduction-based approach to control of DPS is presented. It follows the equation-free paradigm and is therefore appropriate for legacy and microscopic simulators. It relies on the online, adaptive construction of local reduced linear models on the closed-loop trajectory and using them in the context of linear MPC. It is inspired by previously presented control algorithms, building on the notion of equation-free computations (Siettos et al., 2003a; Siettos et al., 2003b; Armaou et al., 2004b; Armaou et al., 2005; Siettos et al., 2006). However, all these approaches employ linearisations at
stationary points. Here, linearisation along a dynamic trajectory is chosen, resulting in a reduced model which could potentially capture more accurately the local dynamic behaviour of the system. Furthermore, in comparison to previous approaches, the proposed algorithm has the advantage of being able to accommodate constraints.

The rest of the Chapter is organized as follows: In Section 5.2 the conventional linear MPC algorithm will be outlined and the Jacobian linearisation strategy of producing linear models for nonlinear systems will be delineated. Next, the model reduction strategy adopted will be analyzed in Section 5.3. The control framework proposed will be introduced in Section 5.4 and an efficient implementation will be presented in Section 5.5. Case studies illustrating its behaviour will be presented in Section 5.6. Finally, a brief summary of the Chapter will be given in Section 5.7.

5.2 Model Predictive Control

Model Predictive Control (MPC) is probably the most popular advanced control strategy, being developed and finding applications both in academia and industry (Camacho and Bordons, 2004). One of its advantages is the ability to handle input, output and state constraints in an intuitive fashion in the context of multivariate control. The basis on which the MPC framework is built around, is that there exists a model for the system under examination, which given initial conditions and inputs, can predict the evolution of state \((x)\) and output \((y)\) variables over a given prediction horizon.

There are many variants of MPC, depending if the model is linear or nonlinear, in continuous or discrete time, if the optimisation problem is solved online, what the objectives of control are (e.g. tracking, economic) and so forth. The common denominator of those approaches is that the control action is computed as the result of an optimisation problem. In the conventional method, the model of the process is in discrete time, in which case the choice of sampling time \((T_s)\) is crucial. The control and prediction horizons are defined as the (integer) number of sampling intervals considered for control and prediction respectively and noted \(N_c\) and \(N_p\). At every sampling interval a set of \(N_c\) control actions are computed, of which only the first is implemented at the next sampling time. A typical
objective of the optimisation problem involved in the MPC is the minimisation of the deviation from a given set-point and the simultaneous minimisation of the control energy required. The set-point can be given from an (static or dynamic) optimisation problem which has been solved offline and has identified the optimal operating conditions.

A linear, discrete-time, state-space model which can be used in MPC, is:

\[
\begin{align*}
    x(k+1) &= A_d x(k) + B_d z(k) \\
    y(k) &= C x(k) + D z(k)
\end{align*}
\]  

(5.1)

Where \( x \in \mathbb{R}^{n_x} \) is the incremental state variable vector: \( x = \tilde{x}_{\text{nom}} - \tilde{x} \), \( \tilde{x} \) is the state vector \( \tilde{x}_{\text{nom}} \) the corresponding vector of nominal states, \( y \in \mathbb{R}^{n_y} \) is the system output, \( z \in \mathbb{R}^{n_z} \) is the system input, \( n_{in} \in \mathbb{N} \) is the number of inputs, \( N \in \mathbb{N} \) is the number of state variables, \( n_{out} \in \mathbb{N} \) is the number of outputs, \( x(k) \) is the state at the \( k \)-th sampling interval, and \( A_d, B_d, C, D \) are the matrices involved in the state space representation of the system \((A_d \in \mathbb{R}^{N \times N}, B_d \in \mathbb{R}^{N \times n_z}, C \in \mathbb{R}^{n_y \times N} \text{ and } D \in \mathbb{R}^{n_y \times n_z})\). The underlying system defined in Equation (5.1) is assumed to be both controllable and observable. This assumption holds if the controllability and observability matrices are full rank. Those matrices are defined respectively as:

\[
C_m = \begin{bmatrix} B_d & A_d B_d & A_d^2 B_d & \ldots & A_d^{N-1} B_d \end{bmatrix}
\]  

(5.2)

and

\[
O_m = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{N-1} \end{bmatrix}
\]  

(5.3)

The optimisation problem solved at a sampling instance \( t \), is

\[
Z = \arg \min_z \left[ \sum_{k=1}^{N_s} \left\| \hat{y}(t + k T_s | t) - r(t + k T_s) \right\|_{L_1}^2 + \sum_{k=1}^{N_s} \left\| z(t + k T_s | t) - z(t + (k-1) T_s | t) \right\|_{L_2}^2 \right]
\]

s.t. \( \hat{x}(t + (k + 1) T_s | t) = A_d \hat{x}(t + k T_s | t) + B_d z(t + k T_s | t) \quad k = 1, 2, \ldots, N_o \)

\[
\hat{y}(t + k T_s | t) = C \hat{x}(t + k T_s | t) + D z(t + k T_s | t)
\]

\[
z_{\text{lower}} \leq z \leq z_{\text{upper}}
\]  

(5.4)
Where $Q_w$ and $R_w$ are weighting matrices of appropriate dimensions, $\hat{x}$, $\hat{y}$ denote predicted values, $(\cdot|t)$ denotes the current time interval $t$, $r$ is the reference signal and $Z = [z(t+1|t)^T \ z(t+2|t)^T \ \ldots \ z(t+N_r|t)^T]^T$ the sequence of future inputs, which satisfies the optimisation problem (5.4). As mentioned above, only the first element of $Z$, $z(t+1|t)$, is implemented and all the rest of the elements are discarded.

If the objective function of the optimisation subproblem is quadratic, such as in (5.4) and the model is linear, resulting in linear constraints, like (5.1) used in (5.4), then the optimisation problem is quadratic can be solved with a conventional quadratic programming method. Formulating QP is more desirable than general NLP problems, as they are convex and hence unimodal. Furthermore, they are solved at a relatively low computational cost, which is crucial for real-time applications. The fact that the QP (5.4) is unimodal does not imply that it is guaranteed to be feasible. In the context of MPC, QP subproblems are known to be produced due to short time horizons (Bemporad et al., 2002). Infeasibility can be handled following the minimal-time or the soft constraint approaches (Mayne et al., 2000).

Obtaining a model of the form of Equation (5.1) is not always a straightforward task, since most systems of engineering interest are nonlinear. A general description of such a model is:

$$
\begin{align*}
\dot{x} &= f(\bar{x}(t),\bar{z}(t),t), \quad \bar{x}(t_0) = \bar{x}_0 \\
\bar{y} &= g(\bar{x}(t),\bar{z}(t),t)
\end{align*}
$$

There are many approaches in using a nonlinear model such as (5.5) for linear control. Some of them were reviewed in Section 2.4; a more comprehensive review focusing on systems identification techniques is provided by Heij et al. (2007). Apart from identification, one possible way of obtaining linear models from nonlinear ones is to perform Jacobian (Lyapunov) linearisation around a nominal point, $\{\bar{x}_{\text{nom}}(t),\bar{z}_{\text{nom}}(t)\}$, which satisfies (5.5) (Datta, 2004; Kwon and Han, 2005). This point is usually chosen to be a stationary point, close to or coinciding with, the set-point. Following this approach, the nominal point is used to define the state space:

$$
\bar{z}(t) = \bar{z}_{\text{nom}} + z(t)
$$
and
\[ \dot{x}(t_0) = \dot{x}_{\text{non}}(t_0) + x(t_0) \]  
(5.7)

for small \( \|x(t_0)\| \) and \( \|u(t)\| \), with \( \|x(t)\| = \sup_t \|x(t)\| \). The requirement for the norms of the deviation variables being small stems from the fact that the linearisation only holds in the neighbourhood of the nominal point.

For small \( \|x\| \) and \( \|y\| \), the states and outputs can be defined as:
\[ \tilde{x}(t) = \tilde{x}_{\text{non}}(t) + x(t) \]  
(5.8)

and
\[ \tilde{y}(t) = \tilde{y}_{\text{non}}(t) + y(t) \]  
(5.9)

A Taylor expansion of the nonlinear system (5.5) using incremental variables and assuming smooth functions \( f \) and \( g \) can be written as:
\[ \begin{align*}
\dot{x}(t) &= \frac{\partial f}{\partial x} \cdot x(t) + \frac{\partial f}{\partial z} \cdot z(t) \\
y(t) &= \frac{\partial g}{\partial x} \cdot x(t) + \frac{\partial g}{\partial z} \cdot z(t)
\end{align*} \]  
(5.10)

Equation (5.10) can be written as:
\[ \begin{align*}
\dot{x}(t) &= A_c x(t) + B_c z(t) \\
y(t) &= C x(t) + D z(t)
\end{align*} \]  
(5.11)

with initial condition \( x(t_0) = x_0 \). The matrices \( A_c \) and \( B_c \) in Equation (5.11) express the Jacobian and the actuator effect and they are given by Equations (5.12) and (5.13) respectively:
\[ A_c = \frac{\partial f}{\partial x} |_{x_{\text{non}}, z_{\text{non}}} \]  
(5.12)

\[ B_c = \frac{\partial f}{\partial z} |_{x_{\text{non}}, z_{\text{non}}} \]  
(5.13)

The linear system (5.11) needs be transformed in discrete time for use within a conventional linear MPC algorithm. It can be shown that \( A_c \) and \( B_c \) for the state space...
model in discrete time (5.1) can be computed from $A_c$ and $B_c$ of the counterpart model in continuous time (5.11), assuming zero-order hold for the input, from the following equations (Franklin et al., 1997):

$$A_v = e^{A_v \tau}$$  
$$B_d = \int_0^{\tau} e^{A_v \tau} B_c d\tau$$

The expression from which the output is computed (and hence matrices $C$ and $D$) remain unaltered by the transformation.

MPControllers are typically implemented using on dedicated computers (Bleris and Kothare, 2005). The trend however is towards system-on-a-chip (SoC) implementations (Bleris et al., 2006). This is due to lower control hardware cost and lower power consumption of SoC implementations. This approach has the added advantage of reduced size of the controller, which is especially important for applications of small size, such as Micro-Electro-Mechanical Systems (MEMS), microfluidic systems and microchemical systems. Large Programmable Logic Arrays (PLC), microcontrollers ($\mu$C), Field Programmable Logic Arrays (FPGA) or Application Specific Integrated Circuits can be used in this context (Johansen et al., 2007; Roldao-Lopes et al., 2009). Implementing MPC on a chip does not imply using control algorithms designed for personal computers, workstations or servers to embedded systems. The latter are known to have very limited computational power, but vast potential for parallel processing. Thus, special control algorithms are required, which exploit characteristics of the architecture of embedded systems. One of the approaches which are compatible for SoC implementation is multi-parametric MPC (mpMPC) (Bemporad et al., 2000; Dua et al., 2008; Pistikopoulos, 2009). The basis for this approach is the explicit calculation of control laws as piecewise affine functions of the states which are calculated offline and tabulated for the controller to use. Hence the online computational cost of the controller is potentially minimal. This approach trades off online computational cost with memory requirements.
5.3. Model reduction

It is always advantageous for the computational cost of control algorithms to be relatively low (Henson, 1998). The desire for efficient algorithms becomes even more crucial and develops into a need since the trend is to implement control algorithms on embedded systems. This is somewhat contradictory with the underlying physical systems becoming more complex. These systems are simulated using complicated models and require controllers with high predictive and controlling capacity. The algorithm presented in this Chapter attempts to accommodate the need for a controller synthesis framework resulting accurate algorithms of low cost.

Model reduction technology is employed in order to render the control algorithms developed computationally efficient and at the same time accurate. Like the optimisation algorithms presented in previous chapters, the property of the underlying systems which is exploited for reduction is dissipation (Alonso and Ydstie, 1996; Brogliato et al., 2007). The approach followed in the work delineated in this Chapter extends the concepts of the “equation-free” notion for control (Siettos et al., 2003a; Siettos et al., 2003b; Armaou et al., 2004b; Armaou et al., 2005; Siettos et al., 2006). This approach is suitable for systems the closed form equations of which are not necessarily known and is inspired by the Recursive Projection Method (Shroff and Keller, 1993). The studies of Kevrekidis and collaborators which were cited above produce low-order models extracted at stationary set-points. The resulting reduced models can be used for linear quadratic control, pole placement and feedback linearisation. In contrast to previous works, here the linearisation is adaptive along the closed-loop trajectory and hence not performed at an equilibrium point.

The basis of this work is an existing timestepper which can simulate the dynamic behaviour of the underlying nonlinear system. This integrator is of the form:

\[ \dot{x}(t_j) = F(\bar{x}, \bar{z}, t_{j+1}) \]  \hspace{1cm} (5.16)

Where \( F(.) \) merely denotes a procedure which is able to predict the evolution of the system at hand subject to a set of initial conditions and inputs; it does not imply knowledge of the governing equations and could possibly only be used in an input/output fashion. Restrictions to accessing the system equations may exist in commercial simulators, legacy codes or microscopic simulators. In the first case the programmer that has developed the code may have only allowed the end user to perform specific tasks with the product rather
than give full access. In the case of legacy codes, the source code may have been lost with
time or may be incompatible with the available compilers; in all those scenarios the tasks
one can use the code for are limited to the early function of the software.

Dissipative systems exhibit a separation of time-scales to fast and slow ones. This
separation results in separation of eigenvalues in the eigenspectrum of the linearised
discrete-time system. Figure 5.1 illustrates such a spectrum. The slow modes correspond to
eigenvalues of high magnitude, i.e. close to or outside of the unit circle. This circle defines
stability, much as the imaginary axis does for steady-state systems: eigenvalues outside the
unit circle correspond to unstable modes. Typically, the number of eigenvalues of high
magnitude is small, whereas the bulk of eigenvalues are close to the origin. Hence, two
cluster of eigenvalues can be seen. The slow modes are the dominant ones, as the fast
modes quickly equilibrate or become enslaved to the slow. The “spectral gap” between the
fast and slow eigenvalues need not be so clear by inspection for the assumption of the
existence of a dominant subspace to hold. Nevertheless, not all systems exhibit those
characteristics. For example the spectra of first-order hyperbolic systems of PDEs are
characterized by a large number of modes close to the unit circle. Applying this kind of
model reduction to such systems would require a large number of modes for the extraction
of an accurate model (Christofides, 2001).

The model reduction approach followed in this work is efficient if the number of dominant
modes, \( m \), is relatively small. Dominant modes can be defined as the ones which are
outside a disk, \( K_\delta \), smaller than the unit circle, the centre of which is the origin and its
diameter is equal to \((1-\delta)\), where \( \delta > 0 \) is a positive parameter.

Without loss of generality, consider the eigenvalues of the system being ordered according
to magnitude:

\[
|\mu_1| \geq \ldots \geq |\mu_m| > 1-\delta \geq |\mu_{m+1}| \geq |\mu_{m+2}| \ldots
\]  

(5.17)

According to the ordering of Equation (5.17) \( \mu_1, \ldots, \mu_m \) lie outside \( K_\delta \) and are the dominant
modes of the system. Let \( P \) denote the maximal invariant subspace of the Jacobian,
corresponding to those modes and \( Q \) its orthogonal complement, so that:

\[
P \oplus Q = \mathbb{R}^n
\]  

(5.18)
Thus, the $\mathbb{H}^N$ space can be written as:

$$\mathbb{H}^N = P\mathbb{H}^N + Q\mathbb{H}^N$$  \hfill (5.19)$$

Where $P$ and $Q$ are projectors onto the subspaces $P$ and $Q$ respectively: $P, Q \in \mathbb{H}^{N,N}$

**Figure 5.1**: Eigenspectrum of a discrete-time system, exhibiting a clear separation of modes.

Equation (5.19) implies that

$$Q = I - P$$  \hfill (5.20)$$

Also $P$ satisfies the following equation, which hold for all projectors:

$$P^2 = P$$  \hfill (5.21)$$

From Equations (5.20) and (5.21), it is easy to show that

$$PQ = 0$$  \hfill (5.22)$$

Any vector $x \in \mathbb{H}^N$ can be decomposed in two unique components: $x = Px + Qx = p + q$, $p \in P$ and $q \in Q$. The “dominance” of the subspace $P$ implies that $Px$ adequately approximates $x$. 
Let $\hat{Z} \in \mathbb{R}^{n \times m}$ an orthonormal basis for $P$ (a basis for $Q$ need not be computed in the context of the proposed scheme). Since the projection onto $P$ is orthogonal and the basis $\hat{Z}$ orthonormal

$$\hat{Z}^T \hat{Z} = I_m$$

(5.23)

Where $I_m$ is an identity matrix of dimension $(m \times m)$.

The projectors $P$ and $Q$ can be computed from:

$$P = \hat{Z} \hat{Z}^T$$

$$Q = I - \hat{Z} \hat{Z}^T$$

(5.24)

The basis for the dominant subspace $P$ can be computed with efficient iterative matrix free methods such as subspace iterations or Arnoldi method. This way the explicit computation of the Jacobian is avoided.

The fast modes may result in the system exhibiting stiffness. Such behaviour is expected if the characteristic time-scale of the system is smaller than the one of the fastest scale (Hadjinicolaou and Goussis, 1998). Thus, applying model reduction strategy which truncates the fast modes may alleviate the stiffness of the original model and prove advantageous.

The sampling time chosen must be large enough for the fast modes to equilibrate or become enslaved to the slow, but also small enough to capture the dynamics of the system. On the other hand, the prediction horizon, which consists of a number of sampling intervals, should be large enough to allow the system to exhibit potential inverse responses and other interesting dynamics, but small enough for the linearisation to hold.

The proposed algorithm neglects the fast modes within each predictive horizon. This may affect the controllers predictive capacity but not the closed loop stability for some classes of systems for some classes of problems (Dubljevic et al., 2006). Some loss of accuracy is compensated by the adaptive nature of the proposed algorithm, following a successive linearisation scheme.
5.4. The proposed predictive control algorithm

The control strategy which is outlined in this section utilizes the model reduction approach described in the previous section for the production of a low-order linear model that can be used in the context of MPC. The novelty of the approach is that an adaptive linearisation on the closed-loop trajectory is performed and is synergetic to the – also adaptive – model reduction. A linearisation around the current state can follow the paradigm of Equation (5.10), in which central role play the Jacobian matrix (Equation (5.12)) and the actuator effect (Equation (5.13)). However, in the proposed algorithm, matrices \( A_c \) and \( B_c \) in (5.10) are low-order. In specific, projections of the original matrices onto the adaptively identified dominant subspace of the system are employed. This way, the full Jacobian of the system, which may be large, need not be computed. The proposed scheme, following the equation-free approach, uses the existing time stepper in order to approximate products of the approximate Jacobian \( F_x \) with given vectors, by employing matrix-free methodologies. Here, \( F_x \) is an approximation of the – defined in Equation (5.12) – Jacobian, obtained using the time stepper. In particular, the time integrator is used with the current state as initial condition and the current value of the inputs, in order to obtain the state vector at a specific reporting horizon, i.e. at a time which differs from the current one by a certain increment, which may be chosen to be equal to the sampling time. According to this paradigm, assuming that \( F(\cdot) \) is the state vector at the end of the reporting time with initial conditions and parameters defined in the arguments of the function, numerical directional perturbations on the Jacobian to the direction of a vector \( v \) can be defined as:

\[
F_x v = \frac{1}{\varepsilon} (F(\tilde{x} + \varepsilon v, \tilde{z}, t) - F(\tilde{x} - \varepsilon v, \tilde{z}, t))
\]  

(5.25)

Where \( \varepsilon \in \mathbb{R}_+ \) is the perturbation size, \( v \in \mathbb{R}^n \) the vector which needs to be multiplied with the Jacobian. The scheme of Equation (5.25) can be considered as the backbone of the matrix-free concept. Here, a central finite difference scheme is followed.

The dominant subspace comprises of the maximal invariant subspace belonging to the \( m \) eigenvalues of \( F_x \) of largest magnitude. This eigenspace can be identified using a matrix-free method. In this work, an implementation of the implicitly restarted Arnoldi method is used (Lehoucq et al., 1998). This procedure results to an orthonormal basis \( \tilde{z} \in \mathbb{R}^{Nm} \), spanning the dominant subspace.
Using the basis \( \hat{Z} \), the state vector \( x \) can be projected onto the dominant subspace in order for a reduced state vector, \( \xi \in \mathbb{R}^m \), to be computed:

\[
\xi = \hat{Z}^T x
\]  

(5.26)

The basis \( \hat{Z} \) is also used for the computation of a reduced Jacobian, \( H \in \mathbb{R}^{m \times m} \):

\[
H = \hat{Z}^T F_z \hat{Z}
\]  

(5.27)

The procedure for evaluating Equation (5.27) resembles the process described in Section 3.5.2 and has roots in the Recursive Projection Method (Shroff and Keller, 1993). The reduced Jacobian, \( H \), formed in two steps. First the \( F_z \hat{Z} \) product is formed and then premultiplied by \( \hat{Z}^T \). The basis \( \hat{Z} \) can be seen as a collection of \( m \) vectors. The column \( j \) of the matrix \( F_z \hat{Z} \) can be computed from Equation (5.25) where the vector \( v \) is the \( j \)-th column of \( \hat{Z} \).

It can be shown, that the Jacobian of the reduced system in continuous time is

\[
A_{c,\text{red}} = I \cdot H
\]  

(5.28)

Matrix \( B_{c,\text{red}} \) in the state-space formulation of the linear system is computed directly with numerical perturbations on the inputs using the time integrator and results from the projection of the matrix of the full system onto the dominant subspace:

\[
B_{c,\text{red}} = \hat{Z}^T B_c
\]  

(5.29)

The linearisation scheme is adaptive, hence matrices \( A_{c,\text{red}} \) and \( B_{c,\text{red}} \) are evaluated at each sampling interval. This way, a reduced counterpart of Equation (5.11) is formed, which describes the evolution of the reduced variables in continuous time:

\[
\dot{\xi}(t) = A_{c,\text{red}} \xi(t) + B_{c,\text{red}} z(t), \quad \xi = \hat{Z}^T x(t) \\
y(t) = C \hat{Z} \xi(t) + Dz(t)
\]  

(5.30)

The system (5.30) needs be transformed to discrete-time, so as to comprise a reduced counterpart of the system of Equation (5.1):
\[
\xi(k + 1) = A\xi(k) + Bz(k) \\
y(k) = C\xi(k) + Dz(k)
\] (5.31)

Matrices \(A\) and \(B\) in Equation (5.31) are recomputed at every sampling interval, \(j\), and should therefore be written as \(A_j\) and \(B_j\). Indices are dropped for simplicity. It can be shown that those matrices can be computed from their continuous time counterparts, \(A_{c,\text{red}}\) and \(B_{c,\text{red}}\), which have already been computed:

\[
A = e^{A_{c,\text{red}}T_j}
\] (5.32)

And

\[
B = \int_0^{T_j} e^{A_{c,\text{red}}\tau} B_{c,\text{red}} d\tau
\] (5.33)

The linearisation procedure described above is the basis of the control strategy proposed. It is condensed in Algorithm 5.1, presented in table 5.1.

**Table 5.1: Algorithm 5.1:** The adaptive linearisation scheme

1. Compute an orthonormal basis \(\hat{Z}\) for the dominant subspace \(P\) of the current state using employing Arnoldi method using directional perturbations on the nonlinear model (5.5) as per Eq. (5.25).

2. Calculate the reduced deviation state variable from Eq. (5.26).

3. Compute the matrices \(A_{c,\text{red}}\) and \(B_{c,\text{red}}\) for the reduced continuous-time state space model (5.30) with numerical directional perturbations to the direction of \(\hat{Z}\).

4. Compute \(A, B, C, D\) for the reduced discrete state space model (5.31) using \(A_{c,\text{red}}\) and \(B_{c,\text{red}}\).

5. Formulate a QP problem using the updated model (5.34).
The low-order linear model of Equation (5.31) can be utilized in the context of an optimisation problem, formulated for each of the sampling intervals, used for the computation of future control actions:

\[
\begin{align*}
Z &= \arg\min_{z} \left[ \sum_{k=1}^{N_p} \left\| \begin{bmatrix} \hat{y}(t+kT_r | t) - r(t+kT_r) \end{bmatrix}^2 + \sum_{k=1}^{N_c} \left\| \begin{bmatrix} z(t+kT_s | t) - z(t+(k-1)T_s | t) \end{bmatrix}^2 \right. \right] \\
&\quad \text{s.t. } \begin{bmatrix} \hat{x}(t+(k+1)T_r | t) = A\hat{x}(t+kT_r | t) + Bz(t+kT_s | t) \quad k = 1, 2, \ldots N_o \\
\hat{y}(t+kT_s | t) = CZ^T \hat{x}(t+kT_r | t) + Dz(t+kT_s | t) \end{bmatrix} \\
&\quad z_{\text{lower}} \leq z \leq z_{\text{upper}} \end{align*}
\]

(5.34)

The QP in (5.34) is reduced and can be considered as the projection of problem (5.4) onto what is considered to be the dominant subspace of the system, which expresses the slow dynamics at the current point of state-space. This optimisation problem will be feasible if the full problem (5.4) is feasible (Corollary 3.3).

The methodology outlined is condensed to Algorithm 5.2. Note that the identification of the local linear model, the formulation of the controller and the solution of the optimisation problem is performed online. The offline procedures it involves are minimal. Initialisation only involves choosing the predictive and control horizons \(N_p\) and \(N_c\), the weights, \(Q_w\) and \(R_w\) as well as the initial conditions for the state, \(\hat{x}\), and manipulated, \(\hat{z}\), variables.

At this point it might be useful to make some remarks regarding the proposed control scheme and the conventional Successive Linearisation approach (SL). First of all, according to the SL scheme, linearisation is performed on a nominal trajectory as opposed to the closed-loop trajectory as proposed in this work. Hence, full state information is required for the reference trajectory. By comparison, here only reference signal for the output variable needs to be given. Also, since the dynamics of the model used for linearisation are closer to the ones observed in the actual system, the number modes that need to be considered as dominant is smaller than the corresponding number for an implementation in which a nominal trajectory is used. SL is typically implemented in multi-step algorithms, which produce a sequence of linear models for each horizon. It results in the formulation of one convex optimisation problem per sampling time, whereas in the general formulation of nonlinear MPC, a general NLP problem is formulated which need not be convex.
Table 5.2: Algorithm 5.2: The proposed control scheme

1. Initialisation: choose horizons $N_p$, $N_c$, weights, $Q_w$ and $R_w$. Also set initial conditions for $\bar{x}$ and $\tilde{z}$

2. At every sampling time:
   
   i. Apply the 1st control decision to the system, computed at the previous time step
   
   ii. Measure the system output and estimate the state
   
   iii. Formulate a new controller by applying Algorithm 5.1
   
   iv. Solve the optimisation subproblem (5.34) and calculate the next $N_c$ control decisions

5.5. A modification for the reduction of computational cost

The most computationally costly step in the Algorithm 5.2 is the formulation of the controller (step 2iii) and in specific the first step of this procedure (step 1, Algorithm 5.1), which is the computation of a basis for the dominant subspace of the system. Matrix-free methods can be utilized, such as Arnoldi method or subspace iterations, which are efficient and suitable for large scale systems. This cost may be reduced by warm-starting those procedures, i.e. by providing initial guesses for a basis of the sought-after subspace. The Arnoldi method which is used in this work requires an initial guess for the eigenvector corresponding to the dominant eigenvalue. It has been shown by Schroff and Keller (1993) that the iterates $\Delta x = x(k) - x(k-1)$ converge to this eigenvector: in the neighbourhood the stationary point, the integrator (5.5) can be seen as performing iterations of the power method (Golub and Van Loan, 1996). Hence, $\Delta x$ is inputted to the Arnoldi procedure as initial guess in the first iteration. In all other iterations, the leading column of the previous basis can be used as initial guess. This column contains the eigenvector corresponding to the eigenvalue of largest magnitude at the time the previous basis was computed.
The computational expense of an iterative eigenanalysis procedure can also in general be significantly reduced by increasing the tolerance, i.e. reducing the accuracy by which the basis for $P$ is computed. Tolerances of the order of $10^{-2}$ have been successfully used. Nevertheless, increasing this tolerance should be performed with caution as it will certainly result in the deterioration of the quality of the local linear model. A safe way of determining the larger tolerance acceptable is to perform a series of offline tests using the underlying model, in order to verify that the bases produced using the increased tolerance do span the eigenspace belonging to the dominant modes at a satisfactory level of accuracy.

Reducing the reporting time chosen is another technique that can potentially be used to reduce the cost of computing $\hat{Z}$. Time integration for a time equal to the value of this parameter takes place every time a Jacobian-vector product needs be computed. Therefore the value of this parameter affects the overall cost significantly. Nevertheless, the choice of reporting time should allow the fast dynamics of the underlying system to become enslaved to the slow ones. Increasing the sampling time is another way of reducing computational cost. That however extends the region in which the nonlinear model is approximated with a linear one in the adaptive linearisation scheme.

A drastic way of reducing the cost of reformulating the control problem, is to only perform this task when necessary, i.e. when the point in state-space significantly differs from the point the previous update of the basis took place, so that one has reasons to suspect that the dynamics of the system may have been altered. Thus, updating of $\hat{Z}$ takes place only if the following condition is met:

$$\frac{\|x_{\text{upd}} - x\|}{\|x_{\text{upd}}\|} \geq \epsilon_{\text{upd}}$$

(5.35)

Where $x_{\text{upd}}$ is the state at the last $\hat{Z}$ update and $\epsilon_{\text{upd}}$ is the heuristically chosen affinity parameter, which expresses a tolerance. Only updating the basis $\hat{Z}$ if Condition (5.35) is met, is based on the assumption that the dominant dynamics of the underlying nonlinear system do not differ significantly for neighbouring points of state space.
Moreover, it is advised to implement a conservative condition, stating that there is a maximum number of sampling intervals, $n_{upd}$, between two consecutive updates of the dominant basis.

$$\quad k - k_{upd} < n_{upd} \quad (5.36)$$

Reformulating the proposed controller, as defined Algorithm 5.2 in conjunction with Conditions (5.35) and (5.36) results in a new Algorithm, outlined in Table 5.3.

**Table 5.3: Algorithm 5.3:** An efficient implementation of the control scheme

1. Initialisation: choose horizons $N_p$, $N_c$, weights, $Q_w$ and $R_w$. Also set initial conditions for $\dot{x}$ and $\dot{z}$

2. For every time interval:
   
   i. Apply the $1^{st}$ control decision to the system, as calculated from the previous time step

   ii. Measure the output of the system and estimate the current state

   iii. *If* conditions (5.35) and (5.36) are met:

   Formulate a new controller by applying Algorithm 5.1

   *Else*:

   use the model from the previous sampling time

   iv. Solve the optimisation subproblem (5.34) and calculate the next $N_c$ control decisions
5.6. Case studies

5.6.1. The base case: control of a tubular reactor

The proposed control algorithms have been applied in two illustrative case studies, in order for their behaviour and efficiency to be explored. Both of those case studies are based on the control of a tubular reactor with recycle, where a first order, elementary irreversible, exothermic reaction takes place: A → B. The model for the reactor consists of two parabolic partial differential equations (PDEs) (Jensen and Ray, 1982; Alonso et al., 2004):

\[
\begin{align*}
\frac{\partial x_1}{\partial t} &= \frac{1}{Pe_1} \frac{\partial^2 x_1}{\partial y^2} - Da \cdot x_1 \exp\left(\gamma x_2 \frac{y}{1 + x_2}\right) \\
\frac{\partial x_2}{\partial t} &= \frac{1}{Pe_2} \frac{\partial^2 x_2}{\partial y^2} + CDa \cdot x_1 \exp\left(\gamma x_2 \frac{y}{1 + x_1}\right) + \beta(x_{2w} - x_2)
\end{align*}
\]

(5.37)

Where \( y \) is the dimensionless longitudinal coordinate, \( x_1 \) is the dimensionless concentration and \( x_2 \) is the dimensionless temperature. \( r \) is the recycle ratio, \( Da \) is the Damköhler number, \( Pe_1 \) is the Pecklet number for mass transport and \( Pe_2 \) for heat transport, \( \beta \) a dimensionless heat transfer coefficient, \( C \) is the dimensionless adiabatic temperature rise, \( x_{2w} \) the dimensionless adiabatic wall temperature, \( \gamma \) the dimensionless activation energy.

The boundary conditions for this model are:

\[
\begin{align*}
\frac{\partial x_1}{\partial y} &= -Pe_1 \left[ (1-r)x_1^0 + r x_1 \big|_{y=1} - r x_1 \big|_{y=0} \right] \\
\frac{\partial x_2}{\partial y} &= -Pe_2 \left[ (1-r)x_2^0 + r x_2 \big|_{y=1} - r x_2 \big|_{y=0} \right]
\end{align*}
\]

(5.38)

at \( y = 0 \) and

\[
\begin{align*}
\frac{\partial x_1}{\partial y} &= 0 \\
\frac{\partial x_2}{\partial y} &= 0
\end{align*}
\]

(5.39)

at \( y = 1 \).

The initial conditions of the system are:
The values for the parameters chosen are: are $Da = 0.1$, $Pe_1 = Pe_2 = 7.0$, $y = 10.0$, $\theta = 2.0$, $C = 2.5$. For the examples presented in Sections 5.6.2 and 5.6.2, the sampling time was 0.01, Arnoldi method was warm-started and the tolerance by which $\hat{Z}$ was computed was $10^{-6}$.

The jacket of the reactor is considered to be comprised of eight heat exchangers of equal surface, the temperatures of which can be controlled independently and comprise the manipulated variables for the control problem. Hence, the dimensionless wall temperature $x_{2w}$ is considered to vary spatially, as a function of the control actions:

$$
x_{2w}(y) = \sum_{j=1}^{8} (H(y - y_{j-1}) - H(y - y_j)) x_j(t)
$$

(5.41)

With $y_i = i / 8$, $i = 0, 1, ..., 8$, $x_{2w}(t)$, $j = 1, 2, ..., 8$, is the dimensionless temperature of the cooling zone for time $t$ and $H(.)$ is the Heaviside step function:

$$
H(y) = \begin{cases} 
0, & y < 0 \\
1, & y \geq 0
\end{cases}
$$

(5.42)

The exit temperature of the reactor was considered to be the output of the system. A schematic of the reactor is given in Figure 5.2.

![Figure 5.2: Schematic representation of a tubular reactor with 8 cooling zones.](image)

Equations (5.37) - (5.39) were semi-discretised using the Finite Element Method. The discretisation is in space over a mesh of 16 nodes, resulting in 32 ODEs. The simulator for the reactor is used by the controller as black box and no access to the equations is
provided. Only state information is exchanged between the time-stepper and the controller, the latter forming a superstructure around the former.

This system exhibits an interesting parametric behaviour: it is stable for no recycle \((r = 0)\) and exhibits a Hopf bifurcation point at \(r = 0.5\). Those two states define a stable system and a system exhibiting sustained oscillations, as illustrated in Figures 5.3 and 5.4.
Figure 5.3: Dynamic profile for the dimensionless (a) temperature and (b) concentration for the open loop reactor behaviour (no recycle).
Figure 5.4: Dynamic profile for the dimensionless (a) temperature and (b) concentration for the open loop reactor with $r = 0.5$. 
5.6.2. Stabilisation of the reactor

The first control scenario chosen is a typical one in control engineering. It can be considered both as stabilization and a tracking problem. A controller is designed so that the oscillating system \( r = 0.5 \) would be effectively stable in terms of the output, i.e. the set point for exit temperature from the reactor is the trajectory of the exit temperature for a stable system without recycle. The manipulated variables are the temperatures of the cooling zones, \( z_j \), which are bounded:

\[ 0 \leq z_j \leq 1, \quad j = 1,2,\ldots,8 \]

(5.43)

The bounds considered are rather tight and were chosen in order to illustrate the ability of the controller to abide by them and achieve the control objective.

A size \( m = 4 \) for the size of the subspace considered to be dominant is adequate in this case. The values for the predictive and control horizons chosen are \( N_p = N_c = 7 \). The sampling time is set to 0.01 (dimensionless).

Both the standard and the efficient algorithms (5.2 and 5.3) have succeeded in fulfilling the control objectives. Figure 5.5a illustrates the output behaviour for the closed loop system, versus the reference trajectory and the open loop oscillating behaviour, whereas Figure 5.5b presents the corresponding input values calculated by the controller of Algorithm 5.2. In Figure 5.6, the closed loop behaviour of the state variables (dimensionless concentration and temperature) is presented. Note that the closed loop system (Figure 5.6) does not follow the same trajectory as the open-loop stable system (Figure 5.3); only the output variable tracks its counterpart.

Algorithm 5.3 also gave similar results, as presented in Figure 5.7. The computational cost of this controller is significantly lower than the previous case. More information on that are provided in Section 5.4.
Figure 5.5: (a) Dynamic behaviour of the open-loop system (dash-dotted line), reference output (dashed line) and closed-loop output (solid line) and (b) the corresponding computed cooling profiles.
Figure 5.6: Dynamic profiles for the dimensionless (a) concentration and (b) temperature throughout the reactor for the closed loop system.
Figure 5.7: (a) Dynamic behaviour of the reference output (dashed line) and the closed loop output (solid line) for the conditional model update case and (b) Corresponding computed cooling zones’ profiles
5.6.3. Destabilisation of the reactor

The second case study presented is based on the stable reactor with no recycle. The problem of tracking control is tackled, in order to exhibit the effectiveness of the controller designed. In specific, the trajectory exit temperature of the reactor for \( r = 0.5 \) acts as a set-point for the system with \( r = 0 \). To render this task possible, both cooling and heating needs be provided from the cooling zones at the jacket of the reactor. Hence, the bounds for the manipulated variables are extended:

\[
-1 \leq z_j \leq 1, \quad j = 1, 2, \ldots, 8
\]

The dynamics of the reference system are more intricate, therefore more modes need be considered as dominant. A dimension \( m = 10 \) for the subspace considered to be dominant is required. Furthermore, longer horizons are required. The values for the predictive and control horizons chosen are \( N_p = N_c = 15 \).

Note that performing linearisation at the stationary point following the conventional Jacobian linearisation approach is not possible in this case, since the system is forced to exhibit sustained oscillations and does not reach a steady-state.

As in the previous case study, both the standard and the efficient algorithms (5.2 and 5.3) have succeeded in fulfilling the control objectives. Figure 5.8a illustrates the output behaviour for the closed loop system, versus the reference trajectory, whereas Figure 5.8b presents the corresponding input values calculated by the controller of Algorithm 5.2. In Figure 5.9, the closed loop behaviour of the state variables (dimensionless concentration and temperature) is presented.

Using Algorithm 5.3 for this case study resulted in a similar behaviour. Figure 5.10a illustrates the output behaviour for the closed loop system, versus the reference trajectory, whereas Figure 5.10b presents the corresponding calculated input values.
5. “Equation-free” linear Model Predictive Control of nonlinear DPS

Figure 5.8: (a) Closed loop output (solid line) and reference (dashed line) for the destabilisation case study and (b) the corresponding calculated inputs
Figure 5.9: The closed loop dynamic profiles for the dimensionless concentration (a) and dimensionless temperature (b) throughout the reactor for the destabilisation case study.
Figure 5.10: (a) The closed loop (solid line) and reference (dashed line) output for the destabilisation case study using Algorithm 5.2. (b) Corresponding computed cooling profiles.
5.6.4. Computational cost

The computational cost of the controllers designed according to the proposed methodologies is reduced by 30 – 80% in comparison to the one of successive linearisation NMPC. Implementations of Algorithm 5.3 are typically 70 to 85% less expensive than implementations of Algorithm 5.2 which is more conservative. The CPU times required for a closed-loop simulation of the two case studies presented in Sections 5.6.2 and 5.6.3 are outlined in Table 5.4. The simulations described were run in MATLAB (R2006b), using single-threaded code and executed on a laptop based on a dual-core Intel® Core™2 Duo T5800 processor (2.00GHz) and 3GB of RAM, running a 32-bit distribution of Microsoft® Windows Vista™. The corresponding source code is presented in the Appendix.

Table 5.4: Computational cost (CPU minutes) of the closed-loop simulations

<table>
<thead>
<tr>
<th>Case study</th>
<th>CPU time with Algorithm 5.2</th>
<th>CPU time with Algorithm 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>“stabilisation” (Section 5.6.2)</td>
<td>26.67</td>
<td>4.79</td>
</tr>
<tr>
<td>“destabilisation” (Section 5.6.3)</td>
<td>16.13</td>
<td>3.99</td>
</tr>
</tbody>
</table>

5.7. Summary

In this Chapter a model reduction-based control strategy for nonlinear distributed parameter systems has been presented. It features an adaptive linearisation approach on the closed-loop trajectory, in order to obtain a local linearisation which is then used in the context of a conventional MPC algorithm. This linearisation only considers the dominant (slow) dynamics of the underlying system, which in turn are identified online from the closed loop trajectory. As a result, the state variables are projections on the original ones onto the low-dimensional subspace. The matrices involved in the state-space description
are updated in every sampling time and are computed efficiently with iterative matrix-free methods in conjunction with numerical directional perturbations. This enables the proposed algorithm to follow the “equation-free” paradigm and hence be suitable for systems the equations of which are not available (e.g. because they are modelled using commercial simulators) or do not exist (e.g. because the underlying system is governed by microscopic rules).

The adaptive nature of the successive linearisation scheme compensates for the nonlinearity of the problem and for neglecting the evolution of the fast modes. Advantages of the proposed algorithm include the formulation of a reduced quadratic subproblem per sampling time, which is convex and unimodal. Its reduced dimension leads to lower computational and memory requirements. This renders the proposed method appropriate for large-scale systems.

An implementation which lowers further the computational cost. Both algorithms have been successfully applied for the control of a tubular reactor with recycle, with various objectives.
6. Conclusions and recommendations for future work

6.1 Conclusions

Optimisation and control are two of the most important fields of Process Systems Engineering, in the sense that they are applied, in one form or another, to all systems belonging to this field. They have been extensively studied and various methods have been proposed to deal with the different classes of systems that arise in practice. However since engineering science evolves, there is always potential and motivation for methodologies which will enable the implementation of better design and operation policies. It is to this end that the work outlined in this thesis was performed.

Optimisation and control conceptually are closely related. Optimisation is performed when designing an industrial system, in order to determine the sizing of the equipment, its type and the materials used. Furthermore, the optimisation procedure suggests the operating conditions that should be used in the process, both based on the initial design and any retrofitting that could potentially be performed later on. These conditions need to satisfy economic operation (expressed by the objective function), and at the same time to ensure safe operation (which often compromises economic benefit to some extent) conforming to the process standards, environmental regulations and/or physical limitations. The latter three factors are expressed in the optimisation problem as constraints.
Provision of setpoints implies a sequential (offline) relationship between optimisation and control procedures. However, in practice most advanced control algorithms are based on optimisation procedures. Such algorithms are Model Predictive Control, Linear Quadratic Regulator, \( H_\infty \) control, \( H_2 \) control and so forth. The objectives of the aforementioned strategies differ, but in all of them, the control action is calculated as a result from the solution of an optimisation subproblem.

The systems on which this work has focused mainly, are large in size (i.e. they involve many equations), and span different temporal and spatial scales. Large-scale systems typically arise from the discretisation of Partial Differential Equations (PDEs) over a computational mesh. It is the separation of timescales which is exploited for model order reduction, in order for optimisation and control tasks to be possible.

Many methodologies focusing on large-scale systems have been proposed over the years. Representative members of the main classes have been presented in Chapter 2. The full model may either be approximated with a surrogate one, which is used for optimisation and control tasks, or can be used in the context of an advanced computational algorithm, which exploits the structure and special properties of the system at hand. Alternatively, a reduced model, linear or nonlinear, or a sequence of reduced models can be computed following a rigorous model order reduction technique. The latter is the approach that has been followed in this work.

Most engineering systems are modelled using commercial simulators, which are known to be accurate. In such cases, there is no guarantee that the simulator provides all the information needed in order to perform optimisation tasks externally. Even if the simulator is in-house developed, the code may not provide such information and its modification is not always possible (due to loss of source code, expired license of suitable compilers, or simply because it is too complicated a task). Conventional black-box methods following the Nested Analysis and Design (NAND) concept may prove inefficient. Thus, algorithms which preserve the non-intrusive nature of NAND techniques while achieving better performance seem appealing. Additional issues arise in cases where closures of the equations, although intuitively possible, may be unavailable. This is the case for systems where microscopic simulators are used standalone, or in conjunction with macroscopic ones, leading to multi-scale models. New algorithms need be devised to account for such cases.

The advantages of the proposed methodologies over conventional techniques are:
6. Conclusions and recommendations for future work

- the ability to handle input/output systems, e.g. systems modelled using commercial simulators or in-house developed legacy solvers
- their extending the use of existing accurate simulators and models, thus building on existing work and knowledge
- their generic nature, exploiting properties exhibited in a broad class of systems, rather than system-specific structures or heuristic approximations
- the adaptive, online identification of a reduced-order approximate model

To recapitulate the contribution of this work to the research community, four algorithms were proposed:

- an algorithm for large-scale, PDE-constrained optimisation problems
- two algorithms for large-scale, PDE-constrained optimisation with inequality constraints and
- an online predictive control framework for large-scale and multi-scale systems.

The behaviour and efficiency of the aforementioned algorithms has been illustrated using a series of case studies which are based on the tubular reactor model (Chapters 3 – 5). Issues that might arise in some specific classes of applications are also addressed.

The optimisation algorithms presented belong to the reduced Hessian class of methods. They follow a 2-step projection scheme onto the subspace of decision variables, rather than the single projection scheme of the reduced Hessian methods. At every iteration, a feasible point is calculated (although an infeasible-path procedure may be followed, as mentioned in Section 3.5.1) and an orthonormal basis for the corresponding dominant subspace is computed with an appropriate matrix-free iterative method. This space is the invariant subspace, which belongs to a certain number of eigenvalues of the linearised system having the algebraically largest real part. This subspace is termed as dominant, as heuristically it has been shown to be a low order approximation of the null space of the system. The eigenspectrum corresponds to the system dynamics, hence the property exploited for model reduction is the separation of timescales, which is manifested as separation of the eigenvalues.

The dominant subspace is used for defining reduced quantities, such as Jacobians, gradients, and estimates for the Lagrange multipliers. The reduced Hessian, which is of size equal to the number of decision variables, belongs to the tangent space of the dominant
subspace. This reduced Hessian is constructed efficiently with numerical directional perturbations and is involved in the solution of a reduced quadratic subproblem, the result of which is a search direction. The dependent and independent variables are updated according to this direction and a new basis is computed to enable the computation of estimates of the adjoint variables for the next iteration.

Updating of the basis for the dominant subspace of the system is typically the most expensive step of the procedure. Instead of performing it twice per iteration, one can heuristically consider computing estimates for the Lagrange multipliers of the next iteration using the basis of the corresponding feasible point, which is computed at the next iteration. This assumption incurs significant reduction in computational cost and does not affect the accuracy by which the optimum is located, as in the last iterations, where the cost of updating the basis is significantly smaller, the unmodified version of the algorithm (with two basis updates per iteration) is employed.

The procedure outlined is appropriate for optimisation problems resulting from the discretisation of the PDEs of the model, which only include equality constraints. In case linear equality constraints are included in the formulation of the model, those can be projected onto the dominant subspace and passed directly to the QP subproblem. However, if nonlinear inequalities are involved, the procedure needs to be modified. There are two approaches that have been considered: using a penalty-function based one and only reducing the system partially.

The penalty function-based approach relies on the use of an overestimator of all constraints. The latter are aggregated using a constraint aggregation function which in this case is a KS function. Subsequently, a new merit function can be defined, which includes the objective and the KS functions. This merit function is further used in the context of the standard algorithm presented. This approach has the advantage of simplicity, but introduces two new parameters, the values of which have to be chosen carefully for the optimisation to be the least conservative possible, but converging to a point simultaneously satisfying the nonlinear inequality constraints.

The second approach followed in order to enable the optimisation scheme to handle nonlinear inequalities can probably be characterized as more rigorous. In specific, the dominant subspace is computed using the linear inequalities as normal and the reduction is
only performed with respect to this null space. The nonlinear inequalities are linearised and passed to the QP subproblem as additional linear inequality constraints.

The control algorithm exploits the separation of timescales for model reduction as well. In this case, the subspace for the low-dimensional dominant subspace of the system is updated in every timestep (conservative version of the algorithm), or whenever a large “jump” in state-space is observed (efficient version). In both cases, the basis for the dominant subspace corresponds to the modes which at the current time instance are identified as dominant and hence would be engaged more significantly than the rest, in the evolution of the system. Moreover, since the modes which are captured in this case are the ones with the largest magnitude, all unsteady modes would be identified and stabilized by the controller.

The dominant subspace for the dynamic system is exploited for the computation of a low-order, linear state-space model, which is suitable for predicting the future behaviour of the system in a neighbourhood of the current time instance. Since this computation is repeated at every time-step, a model reduction-based, successive linearisation scheme is formulated. The recursively projected linearised model resulting from this procedure is used in the context of a conventional Model Predictive Control algorithm, which involves a reduced QP subproblem and outputs a sequence of future control actions. Following the receding horizon notion, only the first element of the sequence is implemented and the rest are discarded.

6.2 Future work

The optimisation methods presented are designed for deterministic nonlinear programming of systems, which only include continuous variables and involve a relatively small number of degrees of freedom. One direction for future work would be to consider mixed integer programming nonlinear programming (MINLP) problems. There is a variety of methods in the literature, in which the solution of such problems is computed by solving a sequence of NLPs. Those methods include the well-known Branch and Bound method (Leyffer, 2001). Other methods, such as Outer-Approximation (Duran and Grossmann,
1986) and Generalized Benders Decomposition (Benders, 1962) involve the solution of a master MILP problem which provides lower bounds and values for the binary variables, which are considered fixed in the nested NLP problems. It appears to be plausible to apply model reduction techniques to optimisation methods that solve continuous relaxations of the original MINLP problems.

There is an evident need to compute the global optimum rather than a local one. The proposed optimiser can be extended to provide for such cases where the objective function is multi-modal. Two alternative routes may be explored. A hybrid algorithm can be designed, along the lines of the work of Wang and Zhang (2007), which employs the “stretching” function technique, i.e. it transforms the objective function to a form which considers previously located minimisers, in conjunction with gradient based methods and simulated annealing. Simulated annealing provides the stochastic forcing for jumps, so that a gradient based method on the transformed objective function can locate the global optimum.

An alternative approach to global optimisation is fully deterministic. The alpha branch and bound (αBB) method, developed by Floudas and co-workers (Maranas and Floudas, 1994; Adjiman et al., 1998) may provide the substrate for deterministic global optimisation. Those methods are based on a convex underestimating scheme in which the original function is decomposed to a sum of terms. In each iteration two steps take place: a branching and a bounding one. Convex underestimators for the functions involved in the NLP are constructed for the determination of a lower bound. The sequence of lower bounds is non-decreasing and thus convergence to the global optimum is achieved. This technique could potentially be combined with the proposed algorithm, but most probably is too intrusive to preserve the black-box nature of the optimiser. Regardless, it would result in the identification of the true global optimum, to which stochastic methods cannot guarantee convergence.

Future applications of the proposed optimisation algorithm include flowsheet optimisation (Ganesh and Biegler, 1987), parameter estimation (Tjoa and Biegler, 1991), dynamic process optimisation, (Biegler et al., 2002) and optimal control (Biros and Ghattas, 2006b).

An extension to the control algorithm presented in Chapter 5 would be to consider the evolution of the slow modes as well, in order to compute approximations to the future behaviour of the state and output variables. The effect of ignoring those terms in the
closed-loop stability is negligible, but in the predictive capability of the algorithm could be potentially significant (Dubljevic and Christofides, 2006). Considering the fast modes in the formulation of the control algorithm would lead to a nonlinear control scheme, which has obvious disadvantages.

An alternative approach to tackling the control problem presented in Chapter 5 would entail the use of the adaptive Trajectory Piecewise Linearisation method (TPWL) on a reference trajectory in order to identify the regions where the nonlinear system can be approximated by a linear one. Then, equation-free model reduction can be performed on the reference trajectory, in order to identify a sequence of reduced order linear models, which would further be used for predictive control. This approach differs significantly from the one presented in Chapter 5, as the linearisation is performed on a reference trajectory, rather than on the real (closed loop) one. This has the added advantage that the algorithm would always be stabilizing, whereas using local linearisations may lead to short-term destabilization. This problem has been mentioned in Chapter 5 and is not really significant, since at the next time step the unstable nodes would be identified and stabilized by the controller. The use of reference trajectory would lead to the possibility of computing the reduced state-space reduced models offline, thus minimizing the computational cost.

The online computational cost can further be reduced by using explicit multi-parametric MPC (Pistikopoulos, 2009) in order to compute control law explicitly as a function of the state and disturbance variables. This formulation has the obvious disadvantage that a reference trajectory needs be provided explicitly and used offline.

Further future work should include applying the proposed algorithm in more realistic case studies. Possible applications include large-scale systems of PDEs such as fuel cells (Tseronis et al., 2008; Bavarian et al., 2010) and polymerization reactors (Zavala and Biegler, 2009a) as well as multi-scale systems, such as microreactors (Hari et al., 2009) and film growth systems (Christofides and Armaou, 2006). The latter are obvious areas of application for the proposed algorithm, due to its equation-free nature, i.e. its ability to handle systems where closures of system equations are unavailable, although they intuitively exist.
References


Biegler LT, Wachter A. SQP SAND strategies that link to existing modeling systems. Large-Scale Pde-Constrained Optimization. 2003;30:199-217


Burnett DS. Finite element analysis: From concepts to applications. Addison-Wesley Reading, MA; 1987.


References


Lehoucq RB, Sorensen DC, Yang C. ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. Siam; 1998.


Lynch S. Dynamical systems with applications using MAPLE. Springer; 2009.


Murtagh BA, Saunders MA. MINOS 5.5 user’s guide. Stanford University; 1998.


Schulz VH. Reduced SQP methods for large-scale optimal control problems in DAE with application to path planning problems for satellite mounted robots: University of Heidelberg; 1996.


References


Appendix: Source code

A.1 FORTRAN Codes corresponding to Chapter 3

A.1.1 1-dof case study, Algorithm 3.1

program main
  implicit none
  integer n, iter, sizebasis, ncv, maxnewtiter, dof
  double precision sstol
  parameter(n=500, sizebasis=10, ncv=20, maxnewtiter=50, dof=1,
            sstol=1e-7)
  integer i, j, ifail
  ! state variables and parameters
  double precision x(n), lambda(dof)
  ! variables for the model reduction procedure
  double precision residhistory(n, sizebasis)
  ! variables for the optimisation procedure
  double precision objfun, gradobjfun(n+dof), pz(dof), a(dof, dof),
  $        istate(dof), obj, ax(dof), flambda(n, dof),
  $        ZBasis(n+dof, dof), srchdirnorm,
  $        Hess(dof, dof), 2t(dof, n+dof),
  $        bl_all(n+dof), bu_all(n+dof), lagrmultnag(dof),
  $        searchdir(n+dof), redgradobjf(dof), lag_mult(n),
  $        norm, ssx(n), Zh(n, sizebasis),
  $        ep, er(sizebasis), ei(sizebasis),
  $        H(sizebasis, sizebasis), temp1(dof, l), temp2,
  $        temp(n+dof, l), pzz(dof, l), temp3, grd(n+dof, l),
  $        lag_mult_red(sizebasis), Zh_prev(n, sizebasis),
  $        timeupd2l, timeupd22

  logical converged
  residhistory = 0.

  open(45, file='optim_path.txt', status='replace')
  write(45,*) 'Iter Parameter_Value'
  write(45,*) '---- ---------------'
  close(45)

  open(86, file='br.txt', status='replace'); close(86)
  open(86, file='pz.txt', status='replace'); close(86)
Appendix: source code

open(86,file='redobjgrad.txt',status='replace'); close(86)
open(78,file='lag_multipliers.txt',status='replace'); close(78)
open(88,file='conv_data.txt',status='replace'); close(88)
open(unit=51,file='timesZupdate.dat',status='replace'); close(51)
call initial_guess(n,dof,x,lamda); lamda=0.1
call eye(dof,Hess)
call bounds(n,dof,bl_all,bu_all)
do i=1,100
if (i.eq.1) then
call ssconv0(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
$             sstol)
ep=1e-3; Zh=residhistory
call cpu_time(timeupdZ1)
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
call cpu_time(timeupdZ2)
timeupdZ2=timeupdZ2-timeupdZ1
open(unit=51,file='timesZupdate.dat',position='append')
write(51,*) i,timeupdZ2; close(51)
call calculate_reduced_Jac(n,dof,x,lamda,Zh,sizebasis,H)
else
call ssconv(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
$             sstol)
ep=5e-2;
call cpu_time(timeupdZ1)
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
call cpu_time(timeupdZ2)
timeupdZ2=timeupdZ2-timeupdZ1
open(unit=51,file='timesZupdate.dat',position='append')
write(51,*) i,timeupdZ2; close(51)
call calculate_reduced_Jac(n,dof,x,lamda,Zh,sizebasis,H)
endif
write(*,*) '***',i,lamda
open(45,file='optim_path.txt',position='append')
write(45,*) 'after newton: lamda=',lamda,x(n),x(n-1),'**end iter',
$        i-1;      close(45)
call obj_func(n,x,objfun)
call obj_func_grad(n,dof,x,gradobjfun)
open(87,file='objgrad.txt'); do j=1,n+dof; write(87,*)
$       gradobjfun(j); enddo; close(87)
call parameter_sensitivity(n,dof,x,lamda,f_lamda)
open(87,file='constr.txt');write(87,*) f_lamda;close(87)
call calc_Z_basis(n,dof,sizebasis,H,Zh,f_lamda,Z_basis)
open(87,file='Z_basis.txt'); do j=1,n+dof;
$       write(87,*) Z_basis(j,dof); enddo; close(87)
if (i.eq.1) then
  !  Zt=transpose(Z_basis); hess=matmul(Zt,Z_basis)
else
call update_hessian(n,dof,Z_basis,lag_mult,hess,x,lamda,
$                lag_mult_red,sizebasis,Zh_prev)
endif
open(86,file='br.txt',position='append'); write(86,*) hess;
Appendix: source code

```fortran
close(86)
call reduced_gradient(n,dof,gradobjfun,Z_basis,redgradobjf)
open(86,file='redobjgrad.txt',position='append');write(86,*)
§  hess;close(86)

call optimise(n,dof,redgradobjf,hess,pz,lagrmultnag,lambda,
  Z_basis,x)
open(86,file='pz.txt',position='append');write(86,*);pz;close(86)

open(78,file='current_x.txt');do j=1,n;write(78,*) x(j);
  enddo;write(78,*);lambda;close(78)

call update_solution(n,dof,pz,Z_basis,x,lambda,srchdirnorm)
call obj_func(n,x,objfun)
call obj_func_grad(n,dof,x,gradobjfun)
open(87,file='objgrad.txt');do j=1,n+dof;write(87,*) gradobjfun(j);
  enddo;close(87)

!---------------------
ep=5e-2;
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lambda,er,ei)
call calculate_reduced_Jac(n,dof,x,lambda,Zh,sizebasis,H)
Zh_prev=Zh
!---------------------

call parameter_sensitivity(n,dof,x,lambda,f_lamda)
open(87,file='constr.txt');write(87,*);f_lamda;close(87)
call calc_Z_basis(n,dof,sizebasis,H,Zh,lambda,Z_basis)
open(87,file='Z_basis.txt');do j=1,n+dof;
  write(87,*) Z_basis(j,dof);
  enddo;close(87)

Zh=transpose(Z_basis);grad(1:n+dof,1)=gradobjfun(1:n+dof);
  temp1=matmul(Zt,grad);
call normeval(dof,temp1,temp3)
pzz(1:dof,1)=pz(1:dof);temp=matmul(Z_basis,pzz);
call normeval(n+dof,temp,temp2)
open(88,file='conv_data.txt',position='append')
write(88,*) i,lambda,objfun,temp3,temp2

Zh_prev=Zh
call lagrange_multiplier(n,dof,gradobjfun,lambda,H,
  Z_basis,lag_mult_red)
open(78,file='lag_multipliers.txt');do j=1,n;
  write(78,*);lag_mult(j);
  enddo;close(78)

write('*** main iteration:',i,'***','lamda=',lambda,'***')
call normeval(n+dof,matmul(Z_basis,pz),temp)
write(45,*);i,lambda,x(n),x(n-1)
write(45,*);objfun,matmul(transpose(Z_basis),gradobjfun),norm
close(45)
open(45,file='optim_path.txt',position='append')
write(45,*);i,lambda,x(n),x(n-1)
write(45,*);objfun,matmul(transpose(Z_basis),gradobjfun),norm
close(45)
open(45,file='optim_data1.txt',position='append')
write(45,*);i,lambda
close(45)
open(45,file='optim_data2.txt',position='append')
write(45,*);i,objfun
close(45)
open(45,file='optim_data3.txt',position='append')
write(45,*);i,matmul(transpose(Z_basis),gradobjfun)
close(45)
open(45,file='optim_data4.txt',position='append')
write(45,*);i,norm
close(45)
```

call sqp_conv_criterion(srchdirnorm,converged)
write(*,*) i,lamda,x(n-1),x(n)
if ((converged).and.(i.gt.1)) exit
enddo
write(*,*) 'Reduced Hessian method converged. lamda=',lamda
end ! of main program

subroutine bounds(n,dof,bl_all,bu_all)
implicit none
integer n,dof,i
double precision bl_all(n+dof),bu_all(n+dof)
do i=1,n/2
  bl_all(2*(i-1)+1)=0
  bl_all(2*(i-1)+2)=0
  bu_all(2*(i-1)+1)=1
  bu_all(2*(i-1)+2)=8
endo
do i=1,dof
  bl_all(n+i)=0
  bu_all(n+i)=.2
endo
end

subroutine obj_func_grad(n,dof,x,gradobjfun)
implicit none
integer n,dof
double precision x(n),gradobjfun(n+dof)
gradobjfun=0.
gradobjfun(n)=-1
end

subroutine obj_func(n,x,objfun)
implicit none
integer n
double precision x(n),objfun
objfun=-x(n)
end

subroutine G_comp(nnodes,nvariables,x,g,Pe1,Pe2,Le,gamma,x2w,Da,
                  dz,b,C)
implicit none
integer nnodes,nvariables,i
double precision x(nnodes*nvariables),G(nnodes*nvariables),Pe1,
                Pe2,Le,gamma,x2w,Da,dz,b,C
do i=3,nnodes*nvariables-2,2
  g(i)=(1/(Pe1*Dz**2))*[(x(i+2)-2*x(i)+x(i-2)-(x(i+2)-x(i-2))/(2*dz))
                      +Da*(1-x(i))*exp(x(i+1)/(1+x(i+1)/gamma))]
endo
g(i)=(1/(Le*Pe2*dz**2))*[(x(i+3)-2*x(i+1)+x(i-1))-(1/(Le*2*dz))*
                        (x(i+3)-x(i-1))-(b/Le)*x(i+1)+C*Da*(1-x(i))*exp(x(i+1)/
                        (1+x(i+1)/gamma))]/b*x2w/Le
endo
g(1)=(1/(Pe1*Dz**2))*(x(3)-2*x(1)+x(3)-2*Pe1*x(1)*dz)-Pe1*x(1)+Da*
subroutine G_drv(n, dof, lamda, x, g)
implicit none
integer nnodes, nvariables, n, dof
double precision Pe1, Pe2, Le, gamma, x2w, b, C, lamda(dof)
parameter(nnodes=250, nvariables=2, Pe1=5., Pe2=5., Le=1., gamma=20.,
$  x2w=0., b=1.5, C=12.)
double precision x(n), g(n), da, dz
da=lamda(1)
if (n .ne. nnodes) stop '*subroutine G_drv: the size of
$ the problem specified is not the same as the one for which functi
$on G can be calculated.'
dz=1./(nnodes-1.)
call G_comp(nnodes, nvariables, x, g, Pe1, Pe2, Le, gamma, x2w, Da, dz, b, C)
end

A.1.2 3-dof case study, Algorithm 3.1

program main
implicit none
integer n, iter, sizebasis, ncv, maxnewtiter, dof
double precision sstol
parameter(n=500, sizebasis=10, ncv=20, maxnewtiter=50, dof=3,
$  sstol=1e-7)
integer i, j, ifail
! state variables and parameters
double precision x(n), lamda(dof)
! variables for the model reduction procedure
double precision residhistory(n, sizebasis)
! variables for the optimisation procedure
double precision objfun, gradobjfun(n+dof), pz(dof), a(dof, dof),
$  istate(dof), obj, ax(dof), f_lamda(n, dof),
$  Z_basis(n+dof, dof), srchdirnorm,
$  Hess(dof, dof), Zt(dof, n+dof),
$  bl_all(n+dof), bu_all(n+dof), lagrmultnag(dof),
$  searchdir(n+dof), redgradobjf(dof), lag_mult(n),
$  norm, ssx(n), Zh(n, sizebasis),
$  ep, er(sizebasis), ei(sizebasis),
$  H(sizebasis, sizebasis), temp1(dof, 1), temp2,
$  temp(n+dof, 1), pzz(dof, 1), temp3, grd(n+dof, 1),
$  lag_mult_red(sizebasis), Zh_prev(n, sizebasis)

logical converged
residhistory=0.
open(45,file='optim_path.txt',status='replace')
write(45,*) 'Iter Parameter_Value'
write(45,*) '----  ---------------'
close(45)
open(45,file='optim_data1.txt',status='replace');close(45)
open(45,file='optim_data2.txt',status='replace');close(45)
open(45,file='optim_data3.txt',status='replace');close(45)
open(45,file='optim_data4.txt',status='replace');close(45)
open(86,file='br.txt',status='replace');close(86)
open(86,file='pz.txt',status='replace');close(86)
open(86,file='redobjgrad.txt',status='replace');close(86)
open(78,file='lag_multipliers.txt',status='replace'); close(78)
open(88,file='conv_data.txt',status='replace');close(88)
call initial_guess(n,dof,x,lamda); lamda=2
call eye(dof,Hess)
call bounds(n,dof,bl_all,bu_all)
do i=1,100
if (i.eq.1) then
call ssconv0(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
$ssto1)
ep=1e-3; Zh=residhistory
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
call calculate_reduced_Jac(n,dof,x,lamda,Zh,sizebasis,H)
else
call ssconv(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
$ssto1)
ep=5e-2;
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
call calculate_reduced_Jac(n,dof,x,lamda,Zh,sizebasis,H)
endif
write(*,*) '***',i,lamda
open(45,file='optim_path.txt',position='append')
write(45,*) 'after newton: lamda=',lamda,x(n),x(n-1),'**end iter',i-1; close(45)
call obj_func(n,x,objfun)
call obj_func_grad(n,dof,x,gradobjfun)
open(87,file='objgrad.txt');do j=1,n+dof; write(87,*)
$ gradobjfun(j); enddo; close(87)
call parameter_sensitivity(n,dof,x,lamda,f_lamda)
open(87,file='constr.txt');write(87,*) 'f_lamda';close(87)
call calc_Z_basis(n,dof,sizebasis,H,Zh,f_lamda,Z_basis)
open(87,file='Z_basis.txt');do j=1,n+dof;
$ write(87,*) Z_basis(j,dof); enddo; close(87)
endif
if (i.eq.1) then
else
call update_hessian(n,dof,Z_basis,lag_mult,hess,x,lamda,
$ lag_mult_red,sizebasis,Z_prev)
endif
open(86,file='br.txt',position='append');write(86,*) hess;
close(86)

call reduced_gradient(n,dof,gradobjfun,Z_basis,redgradobjf)
open(86,file='redobjgrad.txt',position='append');write(86,*)
$hess;close(86)

call optimise(n,dof,redgradobjf,hess,pz,lagmultnag,lambda,
$Z_basis,x)
open(78,file='current_x.txt');do j=1,n;write(78,*) x(j);
$enddo;write(78,*) lambda;close(78)

call update_solution(n,dof,pz,Z_basis,x,lambda,srchdirnorm)
call obj_func(n,x,objfun)

!-------------------
ep=5e-2;
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lambda,er,ei)
call calculate_reduced_Jac(n,dof,x,lambda,Zh,sizebasis,H)
Zh_prev=Zh
!-------------------
call parameter_sensitivity(n,dof,x,lambda,f_lamda)
open(87,file='constr.txt');write(87,*) f_lamda;close(87)
call calc_Z_basis(n,dof,sizebasis,Zh,lambda,f_lamda,2)
open(87,file='Z_basis.txt');do j=1,n+dof;
$write(87,*) Z_basis(j,dof); enddo; close(87)

Zt=transpose(Z_basis);grd(1:n+dof,1)=gradobjfun(1:n+dof);
temp=matmul(Zt,grd);
call normeval(dof,temp1,temp3)
pzz(1:dof,1)=pz(1:dof);temp=matmul(Z_basis,pzz);
call normeval(n+dof,temp,temp2)
open(88,file='conv_data.txt',position='append')
write(88,*) i,lambda,objfun,temp3,temp2
close(88)

Zh_prev=Zh
call lagrange_multiplier(n,dof,gradobjfun,lag_mult,sizebasis,Zh,H,
$lag_mult_red)
open(78,file='lag_multipliers.txt');do j=1,n;
$write(78,*) lag_mult(j); enddo; close(78)

write(45,*) '*** main iteration:',i,'lamda=',lambda,'***
call normeval(n+dof,matmul(Z_basis,pz),norm)
open(45,file='optim_path.txt',position='append')
write(45,*) i,lambda,x(n),x(n-1)
write(45,*) objfun,matmul(transpose(Z_basis),gradobjfun),norm
close(45)
open(45,file='optim_data1.txt',position='append')
write(45,*) lamda; close(45)
open(45,file='optim_data2.txt',position='append')
write(45,*) objfun; close(45)
open(45,file='optim_data3.txt',position='append')
write(45,*) matmul(transpose(Z_basis),gradobjfun); close(45)
open(45,file='optim_data4.txt',position='append')
write(45,*) norm; close(45)
call sqp_conv_criterion(srchdirnorm,converged)
write('**',*) i, lamda, x(n-1), x(n)
if ((converged).and.(i.gt.1)) exit
enddo
write('**',*) 'Reduced Hessian method converged. lamda=', lamda
end ! of main program

subroutine G_drv(n, dof, lamda, x, g)
implicit none
integer n, dof
double precision x(n), lamda(dof), g(n)
call func(n, x, lamda, g)
end

subroutine func(n, x, par, g)  ! equivalent to: g_drv(n, dof, lamda, x, g)
implicit none
integer nn, n, nact, dof, k
parameter (nact = 3, dof=3)
double precision x(n), x1(n/2), x2(n/2), g1(n/2), g2(n/2),
*     dx, g(n), par(dof)
double precision pe1, pe2, da, gamma, b, le, beta, x2w
double precision z, uact(nact)
integer switch(nact), fl1, fl2, fl3

dx = 1.d0 / dble((n/2) - 1)
nn = n / 2
k=0
do i = 1, nn
k=k+1
x1(i) = x(k)
k=k+1
x2(i) = x(k)
end do
*
set parameters values
b= 12.
beta= 1.5d0
pe1= 5.
p2= 5.d0
le= 1.
gamma= 20.
da = 0.1d0

do i=1, nact
uact(i) = par(i)
enddo
switch(1) =1
switch(2) = 0
switch(3) =0
call u_function(n, nact, switch, uact, x2w)
g1(1)= (2.*(x1(2)-(dx*pe1+1.)*x1(1)))/(pe1*dx*dx) - pe1*x1(1)
  + da*(1.-x1(1))*dexp(x2(1)/(1.+(x2(1)/gamma)))
g2(1)= (2.*(x2(2)-(dx*pe2+1.)*x2(1)))/(le*pe2*dx*dx)
  - pe2*x2(1)/le - beta*x2(1)/le
  + (b*da/le)*(1.-x1(1))*dexp(x2(1)/(1.+(x2(1)/gamma)))
  + beta*x2w/le

do i = 2, nn-1
z = dble(i - 1)/dble(nn-1)
if (z.le.1.d0/3.d0) then
  switch (1) = 1
  switch (2) = 0
  switch (3) = 0
  call u_function(n,nact,switch,uact,x2w)
endif
if ((z.gt.1.d0/3.d0).and.(z.le.2.d0/3.d0)) then
  switch (1) = 0
  switch (2) = 1
  switch (3) = 0
  call u_function(n,nact,switch,uact,x2w)
end if
if (z.gt.2.d0/3.d0) then
  switch(1)=0
  switch(2)=0
  switch(3)=1
  call u_function(n,nact,switch,uact,x2w)
endif
g1(i)=(x1(i+1)-2.*x1(i)+x1(i-1))/(pe1*dx*dx)
   1      - (x1(i+1)-x1(i-1))/(2.*dx)
   2      + da*(1.-x1(i))*dexp(x2(i)/(1.+(x2(i)/gamma)))
g2(i)=(x2(i+1)-2.*x2(i)+x2(i+1))/(le*pe2*dx*dx)
   1      - (x2(i+1)-x2(i-1))/(2.*le*dx) - beta*x2(i)/le
   2      + (b*da/le)*(1.-x1(i))*exp(x2(i)/(1.+(x2(i)/gamma)))
   3      + beta*x2w/le
enddo
switch(1) = 0
switch(2) = 0
switch(3) = 1
  call u_function(n,nact,switch,uact,x2w)
g1(nn)=2.*(x1(nn-1)-x1(nn))/(pe1*dx*dx)
   1      + da*(1.-x1(nn))*dexp(x2(nn)/(1.+(x2(nn)/gamma)))
g2(nn)=2.*(x2(nn-1)-x2(nn))/(le*pe2*dx*dx) - beta*x2(nn)/le
   1      + (b*da/le)*(1.-x1(nn))*dexp(x2(nn)/(1.+(x2(nn)/gamma)))
   2      + beta*x2w/le
k=0
do i=1,n/2
  k=k+1
  g(k)=g1(i)
  k=k+1
  g(k)=g2(i)
enddo

subroutine u_function(n,nact,switch,uact,u)
  implicit none
  integer n,nact,j,switch(nact)
  double precision u, UACt(nact)
u=0.d0
do j=1,nact
  u = u + dble(switch(j)) * uact(j)
end do
end

subroutine obj_func_grad(n,dof,x,gradobjfun)
  implicit none
  integer n,dof
  double precision x(n),gradobjfun(n+dof)
  gradobjfun=0.
  gradobjfun(n-1)=1
end
Appendix: source code

subroutine obj_func(n,x,objfun)
implicit none
integer n
double precision x(n),objfun
objfun=-x(n-1)
end

subroutine bounds(n,dof,bl_all,bu_all)
implicit none
integer n,dof,i
double precision bl_all(n+dof),bu_all(n+dof)
do i=1,n/2
bl_all(2*(i-1)+1)=0
bl_all(2*(i-1)+2)=0
bu_all(2*(i-1)+1)=1
bu_all(2*(i-1)+2)=8
enddo
do i=1,dof
bl_all(n+i)=0
bu_all(n+i)=4.
enddo
end

A.1.3 1-dof case study, Algorithm 3.2

program main
implicit none
integer n,iter,sizebasis,ncv,maxnewtiter,dof
double precision sstol
parameter(n=500,sizebasis=10,ncv=20,maxnewtiter=50,dof=1,$
  sstol=1e-7)
integer i,j,ifail
! state variables and parameters
double precision x(n),lamda(dof)
! variables for the model reduction procedure
double precision residhistory(n,sizebasis)
! variables for the optimisation procedure
double precision objfun,gradobjfun(n+dof),pz(dof),a(dof,dof),$
  istate(dof),obj,ax(dof),f_lamda(n,dof),$
  Z_basis(n+dof,dof),archdirnorm,$
  Hess(dof,dof),Zt(dof,n+dof),$
  bl_all(n+dof),bu_all(n+dof),lagrmultnag(dof),$
  searchdir(n+dof),redgradobjf(dof),lag_mult(n),$
  norm,ssx(n),Zh(n,sizebasis),$
  ep,er(sizebasis),ei(sizebasis),$
  H(sizebasis,sizebasis),temp1(dof,1),temp2,$
  temp(n+dof,1),pzz(dof,1),temp3,grd(dof,1),$
  lag_mult_red(sizebasis),Zh_prev(n,sizebasis),$\n  H_prev(sizebasis,sizebasis),x_prev(n),$
  lag_mult_red(n+1),objfun_prev,f_lamda_prev(n,dof),$
  gradobjfun_prev(n+dof),Z_basis_prev(n+dof,dof)$

logical converged,bigstep
residhistory=0.
bigstep=.true.

open(45,file='optim_path.txt',status='replace')
write(45,*)'Iter Parameter_Value'
write(45,*)'---- ---------------'
close(45)

open(86,file='br.txt',status='replace');close(86)
open(86,file='pz.txt',status='replace');close(86)
open(86,file='redobjgrad.txt',status='replace');close(86)
open(78,file='lag_multipliers.txt',status='replace'); close(78)
open(88,file='conv_data.txt',status='replace');close(88)
open(57,file='bigstepbehaviour.txt',status='replace');close(57)
call initial_guess(n,dof,x,lambda); lambda=0.1
call eye(dof,Hess)
call bounds(n,dof,bl_all,hu_all)

do i=1,100
  if (i.eq.1) then
    call ssconv0(n,dof,lambda,x,residhistory,maxnewtiter,sizebasis, $
      sstol)
    ep=1e-3; Zhat=residhistory
    call eb22drv2(n,dof,x,ep,Zhat,ncv,sizebasis,lambda,er,ei)
    call calculate_reduced_Jac(n,dof,x,lambda,Zhat,sizebasis,H)
  else
    call ssconv(n,dof,lambda,x,residhistory,maxnewtiter,sizebasis, $
      sstol)
    ep=5e-2;
    call eb22drv2(n,dof,x,ep,Zhat,ncv,sizebasis,lambda,er,ei)
    call calculate_reduced_Jac(n,dof,x,lambda,Zhat,sizebasis,H)
  endif
  write(*,*)'***',i,lambda
  open(45,file='optim_path.txt',position='append')
  write(45,*)'after newton: lambda=',lambda,x(n),x(n-1),'**end iter', $
    s i-1; close(45)
call obj_func(n,x,objfun)
call obj_func_grad(n,dof,x,gradobjfun)
open(87,file='objgrad.txt');do j=1,n+dof;write(87,*) $
  gradobjfun(j)]; enddo; close(87)
call parameter_sensitivity(n,dof,x,lambda,f_lambda)
open(87,file='constr.txt');write(87,*) f_lambda;close(87)
call calc_Z_basis(n,dof,sizebasis,H,Zhat,f_lambda,Z_basis)
open(87,file='Z_basis.txt');do j=1,n+dof; $
  write(87,*) Z_basis(j,dof); enddo; close(87)
if (i.eq.1) then
  ! Zt=transpose(Z_basis); hess=matmul(Zt,Z_basis)
else
  if (bigstep) then
    
    
  endif
endif
open(57,file='bigstepbehaviout.txt',position='append')
write(57,*) ' step',i,'calculating quantities bef hess upd'
close(57)

call calculate_reduced_Jac(n,dof,x_prev,lamda_prev,Zh,sizebasis,
$                      H_prev)
call obj_func(n,x_prev,objfun_prev)
call obj_func_grad(n,dof,x_prev,gradobjfun_prev)
call parameter_sensitivity(n,dof,x_prev,lamda_prev,f_lamda_prev)
call calc_Z_basis(n,dof,sizebasis,H_prev,Zh,f_lamda_prev,
$                      Z_basis_prev)
!
" termination criterion
Zt=transpose(Z_basis_prev);grd(1:n+dof,1)=gradobjfun_prev(1:n+dof)
templ=matmul(Zt,grd);
call normeval(dof,templ,temp3)
pzz(1:dof,1)=pz(1:dof);temp=matmul(Z_basis_prev,pzz);
call normeval(n+dof,temp,temp2)
open(88,file='conv_data.txt',position='append')
write(88,*) i-1,lamda_prev,objfun_prev,temp3,temp2
close(88)
open(78,file='lag_multipliers.txt');do j=1,n;
$      write(78,*) lag_mult(j); enddo; close(78)
write('(*,*),",i-1,'lamda=',lamda,'****
call normeval(n+dof,matmul(Z_basis_prev,pz),norm)
open(45,file='optim_path.txt',position='append')
write(45,*) i-1,lamda,x(n),x(n-1)
write(45,*) objfun,matmul(transpose(Z_basis_prev),gradobjfun),norm
close(45)
call sqp_conv_criterion(srchdirnorm,converged)
write('(*,*),",i-1,lamda,x(n-1),x(n)
if ((converged).and.(i.gt.1)) exit
!
' termination criterion /end

call lagrange_multiplier(n,dof,gradobjfun_prev,lag_mult,sizebasis,
$                      Zh,H_prev,lag_mult_red)
endif ! bigstep
call update_hessian(n,dof,Z_basis,lag_mult,hess,x,lamda,
$                      lag_mult_red,sizebasis,Zh_prev)
endif
open(86,file='br.txt',position='append');write(86,*) hess;
close(86)
call reduced_gradient(n,dof,gradobjfun,Z_basis,redgradobjf)
open(86,file='redobjgrad.txt',position='append');write(86,*)
$                      hess;close(86)
call optimise(n,dof,gradobjfun,Z_basis,hess,pz,lagrmultnag,lamda,
$                      Z_basis,x)
open(86,file='pz.txt',position='append');write(86,*) pz;close(86)
open(78,file='current_x.txt');do j=1,n;write(78,*) x(j);
$      enddo;write(78,*) lamda;close(78)
call update_solution(n,dof,pz,Z_basis,x,lamda,
$                      srchdirnorm)
if (srchdirnorm.gt.0.5) then
  bigstep=.true.
else
  bigstep=.false.
endif
open(57,file='bigstepbehaviout.txt', position='append')
write(57,*) srchdirnorm, bigstep, i
close(57)

if (bigstep) then
  x_prev=x
  lamda_prev=lamda
  Zh_prev=Zh
else ! bigstep
  open(57,file='bigstepbehaviout.txt', position='append')
  write(57,*) ' step', i, 'continuing after optimisation'
  close(57)
  call obj_func(n,x,objfun)
  call obj_func_grad(n,dof,x,gradobjfun)
  open(87,file='objgrad.txt'); do j=1,n+dof; write(87,*)
      gradobjfun(j); enddo; close(87)

  !----------------------
  ep=5e-2;
  call eb22drv2(n,dof,x,ep,2h,ncv,sizebasis, lamda,er,ei)
  call calculate_reduced_Jac(n,dof,x, lamda,Zh,sizebasis,H)
  Zh_prev=Zh

  !----------------------
  call parameter_sensitivity(n,dof,x, lamda,f_lamda)
  open(87,file='constr.txt'); write(87,*) f_lamda; close(87)
  call calc_Z_basis(n,dof,sizebasis,H,Zh,f_lamda,Z_basis)
  open(87,file='Z_basis.txt'); do j=1,n+dof;
      write(87,*) Z_basis(j,dof); enddo; close(87)
  Zt=transpose(Z_basis); grd(1:n+dof,1)=gradobjfun(1:n+dof);
  temp=matmul(Zt,grd);
  call normeval(dof,temp,temp3)
  pzz(1:dof,1)=pz(1:dof); temp=matmul(Z_basis,pzz);
  call normeval(n+dof,temp,temp2)
  open(88,file='conv_data.txt', position='append')
  write(88,*) i, lamda, objfun, temp3,temp2
  close(88)

  Zh_prev=Zh
  call lagrange_multiplier(n,dof, gradobjfun, lag_mult, sizebasis, Zh,H, 
  lag_mult_red)
  open(78,file='lag_multipliers.txt'); do j=1,n;
      write(78,*) lag_mult(j); enddo; close(78)
endif !bigstep
write(*,*) '*** main iteration:', i, 'lamda=', lamda, '***'
write(*,*) 'lag_mult_red' 
write(45,*) i, lamda, x(n), x(n-1)
write(45,*) objfun, matmul(transpose(Z_basis),gradobjfun), norm
close(45)
open(45,file='optim_data1.txt', position='append')
write(45,*) i, lamda; close(45)
open(45,file='optim_data2.txt', position='append')
write(45,*) i, objfun; close(45)
open(45,file='optim_data3.txt', position='append')
write(45,*) i, matmul(transpose(Z_basis),gradobjfun); close(45)
open(45, file='optim_data4.txt', position='append')
write(45,*) i, norm; close(45)

call sqp_conv_criterion(srchdirnorm, converged)
write(*,*) i, lambda, x(n-1), x(n)
if ((converged).and.(i.gt.1)) exit
enddo
write(*,*) 'Reduced Hessian method converged. lambda=', lambda
end ! of main program

subroutine bounds(n, dof, bl_all, bu_all)
implicit none
integer n, dof, i
double precision bl_all(n+dof), bu_all(n+dof)
do i=1, n/2
   bl_all(2*(i-1)+1)=0
   bl_all(2*(i-1)+2)=0
   bu_all(2*(i-1)+1)=1
   bu_all(2*(i-1)+2)=8
endo
do i=1, dof
   bl_all(n+i)=0
   bu_all(n+i)=.2
endo
end

subroutine obj_func_grad(n, dof, x, gradobjfun)
implicit none
integer n, dof
double precision x(n), gradobjfun(n+dof)
gradobjfun=0.
gradobjfun(n)=-1
end

subroutine obj_func(n, x, objfun)
implicit none
integer n
double precision x(n), objfun
objfun=-x(n)
end

subroutine G_comp(nnodes, nvariables, x, g, Pe1, Pe2, Le, gamma, x2w, Da, dz, b, C)
implicit none
integer nnodes, nvariables, i
double precision x(nnodes*nvariables), g(nnodes*nvariables), Pe1,
         G(nnodes*nvariables), Pe2, Le, gamma, x2w, Da, dz, b, C
do i=3, nnodes*nvariables-2, 2
   g(i)=(1/(Pe1*Dz**2))*(x(i+2)-2*x(i)+x(i-2))-(x(i+2)-x(i-2))/(2*dz)
   +Da*(1-x(i))*exp((x(i+1)+x(i-1))/(2*gamma))/((x(i)+x(i+1))/gamma)
endo
$g(i+1) = (1/(Le*Pe2*dz**2)) * (x(i+3) - 2*x(i+1) + x(i-1)) - (1/(Le*2*dz)) *$
$ (1*x(i+1) + C*Da*(1-x(i)) * exp(x(i+1)/$
$ (1+x(i+1)/gamma)) + b*x2w/Le$
$enddo$
g(1) = (1/(Pe1*Dz**2)) * (x(3) - 2*x(1) + x(3) - 2*Pe1*x(1)*dz) - Pe1*x(1) + Da *$
$ (1-x(1)) * exp(x(2)/(1+x(2)/gamma))$
g(2) = (1/(Le*Pe2*dz**2)) * (x(4) - 2*x(2) + (x(4) - 2*Pe2*x(2)*dz) - Pe2 *$
$ x(2)/Le - (b/Le)*x(2) + C*Da*(1-x(1)) * exp(x(2)/(1+x(2)/gamma)) +$
$ b*x2w/Le$
g(nnodes*nvariables-1) = (1/(Pe1*Dz**2)) * (x(nnodes*nvariables-3) - 2*x(nnodes*nvariables-3)) -$
$ (x(nnodes*nvariables-3) - x(nnodes*nvariables-3)) / (2*dz) + Da + (1-x(nnodes*nvariables-1)) *$
$ exp(x(nnodes*nvariables) / (1+x(nnodes*nvariables)/gamma))$
g(nnodes*nvariables) = (1/(Le*Pe2*dz**2)) * (x(nnodes*nvariables-2) -$
$ 2*x(nnodes*nvariables) + x(nnodes*nvariables-2) - (1/(Le*2*dz))) -$
$ (x(nnodes*nvariables-2) - x(nnodes*nvariables-2)) - (b/Le)*$
$ x(nnodes*nvariables) + C*Da*(1-x(nnodes*nvariables-1)) *$
$ exp(x(nnodes*nvariables) / (1+x(nnodes*nvariables)/gamma)) + b*x2w/Le$
$end$

subroutine G_drv(n,dof,lamda,x,g)$
$implicit none$
$integer nnodes,nvariables,n,dof$
$double precision Pe1,Pe2,Le,gamma,x2w,b,C,lamda(dof)$
$parameter(nnodes=250,nvariables=2,Pe1=5.,Pe2=5.,Le=1.,gamma=20.,$
$ x2w=0.,b=1.5,C=12.)$
$double precision x(n),g(n),da,dz$
$da=lamda(1)$
$if (n.ne.(nnodes*nvariables)) stop 'subroutine G_drv: the size of$
$ the problem specified is not the same as the one for which functi$
$ $on G can be calculated.'$
$dz=1./(nnodes-1.)$
$call G_comp(nnodes,nvariables,x,g,Pe1,Pe2,Le,gamma,x2w,Da,dz,b,C)$
$end$

A.1.4 3-dof case study, Algorithm 3.2

program main$
$implicit none$
$integer n,iter,sizebasis,ncv,maxnewtiter,dof$
$double precision sstol$
$parameter(n=500,sizebasis=10,ncv=20,maxnewtiter=50,dof=3,$
$ sstol=1e-7)$
$integer i,j,ifail$
$! state variables and parameters$
$double precision x(n),lamda(dof)$
$! variables for the model reduction procedure$
$double precision residhistory(n,sizebasis)$
$! variables for the optimisation procedure$
$double precision objfun,gradobjfun(n+dof),pz(dof),a(dof,dof),$
$ istate(dof),obj_ax(dof),f_lamda(n,dof),$
$ Z_basis(n+dof,dof),srchdirnorm,$
$ Hess(dof,dof),Zt(dof,n+dof),$
$ bl_all(n+dof),bu_all(n+dof),lagrmultnag(dof),$}
$ searchdir(n+dof),redgradobjf(dof),lag_mult(n),$
$ norm,ssx(n),Zh(n,sizebasis),$
$ ep,er(sizebasis),ei(sizebasis),$}
$ H(sizebasis,sizebasis),temp1(dof,1),temp2,$}
$ temp(n+dof,1),pz(dof,1),temp3,grd(n+dof,1),$}
$
Appendix: source code

$ lag_mult_red(sizebasis), Zh_prev(n, sizebasis),
$ H_prev(sizebasis, sizebasis), x_prev(n),
$ lamda_prev(dof), objfun_prev, f_lamda_prev(n, dof),
$ gradobjfun_prev(n+dof), Z_basis_prev(n+dof, dof)

logical converged, bigstep

residhistory=0.

open(45, file='optim_path.txt', status='replace')
write(45,*) 'Iter  Parameter_Value'
write(45,*) '----  ---------------'
close(45)

open(45, file='optim_data1.txt', status='replace'); close(45)
open(45, file='optim_data2.txt', status='replace'); close(45)
open(45, file='optim_data3.txt', status='replace'); close(45)
open(45, file='optim_data4.txt', status='replace'); close(45)

open(86, file='br.txt', status='replace'); close(86)
open(86, file='pz.txt', status='replace'); close(86)
open(86, file='redobjgrad.txt', status='replace'); close(86)
open(87, file='lag_multipliers.txt', status='replace'); close(87)
open(87, file='conv_data.txt', status='replace'); close(87)

call initial_guess(n, dof, x, lamda); lamda=2

call eye(dof, Hess)

call bounds(n, dof, bl_all, bu_all)

do i=1,100
if (i.eq.1) then
  call ssconv0(n, dof, lamda, x, residhistory, maxnewtiter, sizebasis, $
              sstol)
  ep=1e-3; Zh=residhistory
  call eb22drv2(n, dof, x, ep, Zh, ncv, sizebasis, lamda, er, ei)
  call calculate_reduced_Jac(n, dof, x, lamda, Zh, sizebasis, H)
else
  call ssconv(n, dof, lamda, x, residhistory, maxnewtiter, sizebasis, $
              sstol)
  ep=5e-2;
  call eb22drv2(n, dof, x, ep, Zh, ncv, sizebasis, lamda, er, ei)
  call calculate_reduced_Jac(n, dof, x, lamda, Zh, sizebasis, H)
endif
write(*,*) '***', i, lamda

open(45, file='optim_path.txt', position='append')
write(45,*) 'after newton: lamda=', lamda, x(n), x(n-1), '**end iter', $
             i-1;      close(45)

call obj_func(n, x, objfun)
call obj_func_grad(n, dof, x, gradobjfun)
open(87, file='objgrad.txt'); do j=1, n+dof; write(87,*) gradobjfun(j); endo; close(87)
call parameter_sensitivity(n, dof, x, lamda, f_lamda)
open(87, file='constr.txt'); write(87,*) f_lamda; close(87)
call calc_Z_basis(n, dof, sizebasis, H, Zh, f_lamda, Z_basis)
open(87, file='Z_basis.txt'); do j=1, n+dof;
Appendix: source code

$       write(87,*) Z_basis(j,dof); enddo; close(87)

if (i.eq.1) then
  !
  Zt=transpose(Z_basis); hess=matmul(Zt,Z_basis)
else

if (bigstep) then
  open(57,file='bigstepbehaviout.txt',position='append')
  write(57,*) ' step',i,'calculating quantities bef hess upd'
  close(57)
  call calculate_reduced_Jac(n,dof,x_prev,lamda_prev,Zh,sizebasis,$
                           H_prev)
  call obj_func(n,x_prev,objfun_prev)
  call obj_func_grad(n,dof,x_prev,gradobjfun_prev)
  call parameter_sensitivity(n,dof,x_prev,lamda_prev,f_lamda_prev)
  call calc_Z_basis(n,dof,sizebasis,H_prev,Zh,f_lamda_prev,$
                      Z_basis_prev)
  !     termination criterion
  Zt=transpose(Z_basis_prev);grd(1:n+dof,1)=gradobjfun_prev(1:n+dof)
  temp1=matmul(Zt,grd);
  call normeval(dof,temp1,temp3)
  pzz(1:dof,1)=pz(1:dof);temp=matmul(transpose(Z_basis_prev),pzz);
  open(88,file='conv_data.txt',position='append')
  write(88,*) i-1,lamda_prev,objfun_prev,temp3,temp2
  close(88)
  open(78,file='lag_multipliers.txt');do j=1,n;
    write(78,*) lag_mult(j); enddo; close(78)
  write(*,*) '*** main iteration:',i-1,'lamda=',lamda,'***'
  call normeval(n+dof,matmul(Z_basis_prev,pz),norm)
  open(45,file='optim_path.txt',position='append')
  write(45,*) i-1,lamda,x(n),x(n-1)
  write(45,*) objfun,matmul(transpose(Z_basis_prev),gradobjfun),norm
  close(45)
  call sqp_conv_criterion(srchdirnorm,converged)
  write(*,*) i-1,lamda, x(n-1), x(n)
  if ((converged).and.(i.gt.1)) exit
  !     termination criterion /end
endif ! bigstep

call update_hessian(n,dof,Z_basis,lag_mult,hess,x,lamda,$
                     lag_mult_red,sizebasis,Zh_prev)
endif

open(86,file='br.txt',position='append');write(86,*) hess;close(86)

call reduced_gradient(n,dof,gradobjfun,Z_basis,redgradobjf)
open(86,file='redobjgrad.txt',position='append');write(86,*)
$                  hess;close(86)

call optimise(n,dof,redgradobjf,hess,pz,lagrmultnag,lamda,$
              Z_basis,x)
open(86,file='pz.txt',position='append');write(86,*) pz;close(86)

open(78,file='current_x.txt');do j=1,n;write(78,*) x(j);
Appendix: source code

enddo;write(78,*) lamda;close(78)

call update_solution(n,dof,pz,Z_basis,x,lamda,srchdirnorm)

if (srchdirnorm.gt.0.5) then
    bigstep=.true.
else
    bigstep=.false.
endif

open(57,file='bigstepbehaviout.txt',position='append')
write(57,*) srchdirnorm,bigstep,i
close(57)

if (bigstep) then
    x_prev=x
    lamda_prev=lamda
    Zh_prev=Zh
else ! bigstep
    open(57,file='bigstepbehaviout.txt',position='append')
    write(57,*) ' step',i,'continuing after optimisation'
    close(57)
endif ! bigstep

call obj_func(n,x,objfun)
call obj_func_grad(n,dof,x,gradobjfun)
open(87,file='objgrad.txt');do j=1,n+dof; write(87,*)
$       gradobjfun(j); enddo; close(87)

!--------------------
ep=5e-2;
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
call calculate_reduced_jac(n,dof,x,lamda,Zh,sizebasis,H)
Zh_prev=Zh
!--------------------
call parameter_sensitivity(n,dof,x,lamda,f_lamda)
open(87,file='constr.txt');write(87,*) f_lamda;close(87)
call calc_Z_basis(n,dof,sizebasis,H,Zh,f_lamda,Z_basis)
open(87,file='Z_basis.txt');do j=1,n+dof;
$       write(87,*) Z_basis(j,dof); enddo; close(87)

Zt=transpose(Z_basis);grd(1:n+dof,1)=gradobjfun(1:n+dof);
temp1=matmul(Zt,grd);
call normeval(dof,temp1,temp3)
pzz(1:dof,1)=pz(1:dof);temp=matmul(Z_basis,pzz);
call normeval(n+dof,temp,temp2)
open(88,file='conv_data.txt',position='append')
write(88,*) i,lamda,objfun,temp3,temp2
close(88)

Zh_prev=Zh
call lagrange_multiplier(n,dof,gradobjfun,lag_mult,sizebasis,Zh,H,$
$                         lag_mult_red)
open(78,file='lag_multipliers.txt');do j=1,n;
$        write(78,*) lag_mult(j); enddo; close(78)

endif ! bigstep
write(*,*) '*** main iteration:',i,'lamda=',lamda,'***
call normeval(n+dof,matmul(Z_basis,pz),norm)
open(45,file='optim_path.txt',position='append')
write(45,*),i,lamda,x(n),x(n-1)
write(45,*),objfun,matmul(transpose(Z_basis),gradobjfun),norm
close(45)
open(45,file='optim_data1.txt',position='append')
write(45,*),i,lamda; close(45)
open(45,file='optim_data2.txt',position='append')
write(45,*),i,objfun; close(45)
open(45,file='optim_data3.txt',position='append')
write(45,*),i,matmul(transpose(Z_basis),gradobjfun); close(45)
open(45,file='optim_data4.txt',position='append')
write(45,*),i,norm; close(45)
call sqp_conv_criterion(srchdirnorm,converged)
write(*,*) i,lamda,x(n-1),x(n)
if ((converged).and.(i.gt.1)) exit
enddo
write(*,*) 'Reduced Hessian method converged. lamda=',lamda
end ! of main program

subroutine obj_func_grad(n,dof,x,gradobjfun)
implicit none
integer n,dof
double precision x(n),gradobjfun(n+dof)
gradobjfun=0.
gradobjfun(n-1)=-1
end

subroutine obj_func(n,x,objfun)
implicit none
integer n
double precision x(n),objfun
objfun=-x(n-1)
end

subroutine bounds(n,dof,bl_all,bu_all)
implicit none
integer n,dof,i
double precision bl_all(n+dof),bu_all(n+dof)
do i=1,n/2
   bl_all(2*(i-1)+1)=0
   bl_all(2*(i-1)+2)=0
   bu_all(2*(i-1)+1)=1
   bu_all(2*(i-1)+2)=8
endo
do i=1,dof
   bl_all(n+i)=0
   bu_all(n+i)=4.
endo
end
appendix: source code

subroutine G_drv(n,dof,lamda,x,g)
implicit none
integer n,dof
double precision x(n),lamda(dof),g(n)
call func(n,x,lamda,g)
end

subroutine func(n,x,par,g)  ! equivalent to: g_drv(n,dof,lamda,x,g)
implicit none
integer nn,n,i,nact, dof,k
parameter (nact = 3,dof=3)
double precision x(n),x1(n/2),x2(n/2),gl(n/2),g2(n/2),
*   dx,g(n),par(dof)
double precision pe1,pe2,da, gamma,b,le,beta,x2w
double precision z,uact(nact)
integer switch(nact),fl1,fl2,fl3
dx = 1.d0 / dble((n/2) - 1)
nn = n / 2
k=0
do i = 1,nn
  k=k+1
  x1(i) = x(k)
  k=k+1
  x2(i) = x(k)
end do
*      set parameters values
  b=  12.
beta=   1.5d0
pe1=   5.
pe2=   5.d0
le=  1.
gamma=   20.
da = 0.1d0
do i=1, nact
  uact(i) = par(i)
endo
do switch(1) =1
switch(2)= 0
switch(3) =0
call u_function(n,nact,switch,uact,x2w)
g1(l)={2.*(x1(2)-(dx*pe1+1.)*x1(1))}/(pe1*dx*dx) - pe1*x1(1)
1      + da*(1.-x1(1))*dexp(x2(1)/(1.+(x2(1)/gamma)))
g2(l)={2.*(x2(2)-(dx*pe2+1.)*x2(1))}/(le*pe2*dx*dx)
1      - pe2*x2(1)/le - beta*x2(1)/le
2      + (b*da/le)*(1.-x1(1))*dexp(x2(1)/(1.+(x2(1)/gamma)))
3      + beta*x2w/le
do i = 2,nn-1
  z = dble(i - 1)/dble(nn-1)
  if (z.le.1.d0/3.d0) then
    switch (1) = 1
    switch (2) = 0
    switch (3) = 0
    call u_function(n,nact,switch,uact,x2w)
  endif
  if ((z.gt.1.d0/3.d0).and.(z.le.2.d0/3.d0)) then
    switch (1) = 0
    switch (2) = 1
    switch (3) = 0
    call u_function(n,nact,switch,uact,x2w)
  end if
  if (z.gt.2.d0/3.d0) then
    switch(1)=0
  endif

switch(2)=0
switch(3)=1
call u_function(n,nact,switch,uact,x2w)
endif

g1(i)=\frac{x1(i+1)-2*x1(i)+x1(i-1)}{(pe1*dx*dx)}
1 - \frac{x1(i+1)-x1(i-1)}{(2*dx)}
2 + da*(1.-x1(i))*dexp(x2(i)/(1.+(x2(i)/gamma)))
\frac{g2(i)=\frac{x2(i+1)-2*x2(i)+x2(i+1)}{(pe2*dx*dx)}
1 - \frac{x2(i+1)-x2(i-1)}{(2*dx)} - beta*x2(i)/le
2 + (b*da/le)*(1.-x1(i))*dexp(x2(i)/(1.+(x2(i)/gamma)))
3 + beta*x2w/le

switch(1) = 0
switch(2) = 0
switch(3) = 1
call u_function(n,nact,switch,uact,x2w)
g1(nn)=\frac{2*(x1(nn-1)-x1(nn))}{(pe1*dx*dx)}
1 + da*(1.-x1(nn))*dexp(x2(nn)/(1.+(x2(nn)/gamma)))
g2(nn)=\frac{2*(x2(nn-1)-x2(nn))}{(pe2*dx*dx)} - beta*x2(nn)/le
1 + (b*da/le)*(1.-x1(nn))*dexp(x2(nn)/(1.+(x2(nn)/gamma)))
2 + beta*x2w/le

subroutine u_function(n,nact,switch,uact,u)
imPLICIT none
INTEGER n,nact,j,switch(nact)
DOUBLE PRECISION u, UAct(nact)
u=0.d0
do j=1,nact
u = u + dble(switch(j)) * uact(j)
end do
end subroutine u_function(n,nact,switch,uact,u)

A.1.5 Utilities: subroutines used in the codes of Sections A.1.1 – A.1.4

subroutine update_hessian(n,dof,ZBasis,lag_mult,hess,x,lamda,lag_mult_red,sizebasis,Zh)
imPLICIT none
INTEGER n,dof,i,j,k,one,zero,sizebasis
PARAMETER(one=1,zero=0)
DOUBLE PRECISION x(n), ZBasis(n+dof,dof), Hess(dof,dof),
$ lag_mult(n), xfull(n+dof), lamda(dof), h,$
$ fullhess(dof,dof,n), sstol, objfun1, objfun2,$
$ objfun3, objfun4, hessobjf(dof,dof),$
$ residhistory(n,one), zh(n,sizebasis),$
$ zhT(sizebasis,n), lag_mult_red(sizebasis),$
$ temp_n(n,one)
DOUBLE PRECISION,dimension(n+dof)::xfull1,xfull2,xfull3,xfull4
DOUBLE PRECISION,dimension(n)::f1,f2,f3,f4
h=1e-3
sstol=1e-7
xfull(1:n)=x(1:n); xfull(n+1:n+dof)=lamda(1:dof)
zhT=transpose(zh)
do i=1,dof
  do j=1,dof
    do k=1,n+dof
      xfull1(k)=xfull(k)+h*Z_basis(k,i)+h*Z_basis(k,j)
      xfull2(k)=xfull(k)+h*Z_basis(k,i)-h*Z_basis(k,j)
      xfull3(k)=xfull(k)-h*Z_basis(k,i)+h*Z_basis(k,j)
      xfull4(k)=xfull(k)-h*Z_basis(k,i)-h*Z_basis(k,j)
    enddo
    call G_drv(n,dof,xfull1(n+1:n+dof),xfull1(1:n),f1)
    call G_drv(n,dof,xfull2(n+1:n+dof),xfull2(1:n),f2)
    call G_drv(n,dof,xfull3(n+1:n+dof),xfull3(1:n),f3)
    call G_drv(n,dof,xfull4(n+1:n+dof),xfull4(1:n),f4)
    do k=1,n
      fullhess(i,j,k)=(f1(k)-f2(k)-f3(k)+f4(k))/((2*h)**2)
    enddo
  enddo
enddo

do i=1,dof
  do j=1,dof
    do k=1,n
      fullhess(i,j,k)=(f1(k)-f2(k)-f3(k)+f4(k))/((2*h)**2)
    enddo
  enddo
enddo

do i=1,dof
  do j=1,dof
    do k=1,n
      fullhess(i,j,k)=(f1(k)-f2(k)-f3(k)+f4(k))/((2*h)**2)
    enddo
  enddo
enddo

end

subroutine lagrange_multiplier(n,dof,gradobjfun,lag_mult, sizebasis,Zh,H,lag_mult_red)
!
imPLICIT none
integer n,dof,ipiv(n),info,lwork,i,nrhs,sizebasis
parameter(nrhs=1)
double precision gradobjfun(n+dof),
  lag_mult(n),temp_n(n,nrhs),temp_all(n+dof,nrhs),
  work(n),Zh(n,sizebasis),lag_mult_red(sizebasis),
  H(sizebasis,sizebasis),temp(sizebasis,nrhs),
  Zht(sizebasis,n),invH(sizebasis,sizebasis),
  temp_backproj(n,nrhs),temp_graddof(n,nrhs),
  lag_mult_red1(sizebasis),lag_mult1(n),norm
!
ZhT=transpose(Zh)
!
lwork=n
invH=transpose(H)
do i=1,n
  temp_graddof(i,1)=gradobjfun(i)
enddo
!
temp=matmul(Zht,temp_graddof)
temp=-temp
call dgesv(sizebasis,nrhs,invH,sizebasis,ipiv,temp,sizebasis,info)
if (info.ne.0) stop 'error from dgesv in lagr.multipl. calculation'

temp_backproj=matmul(Zh,temp)
do i=1,sizebasis
lag_mult_red(i)=temp(i,1)
enddo
do i=1,n
lag_mult(i)=temp_backproj(i,1)
enddo

open(78,file='lag_multipliers.txt',position='append')
do i=1,n; write(78,*) lag_mult(i); enddo;write(78,*)'----------'
close(78)
end

subroutine sqp_conv_criterion(srchdirnorm,converged)
implicit none
double precision srchdirnorm
logical converged
converged=.false.
if (srchdirnorm.le.1e-4) converged=.true.
end

subroutine update_solution(n,dof,pz,Z_basis,x,lamda,srchdirnorm)
implicit none
integer n,dof,i
double precision pz(dof),Z_basis(n+dof,dof),x(n),lamda(dof),
$                 srchdirnorm,temp(dof,1),temp_all(n+dof,1),$
$                 searchdir(n+dof)$
do i=1,dof
   temp(i,1)=pz(i)
enddo
temp_all=matmul(Z_basis,temp)
do i=1,n+dof
   searchdir(i)=temp_all(i,1)
enddo
do i=1,n
   x(i)=x(i)+searchdir(i)
enddo
   lamda(1:dof)=lamda(1:dof)+searchdir(n+1:n+dof)
call normeval(n+dof,searchdir,srchdirnorm)
end

subroutine optimise(n,dof,redgradobjf,hess,pz,lagrmultnag,
$                 lamda,Z_basis,x)$
implicit none
integer n,dof,ifail,istate(n+2*dof),iter,nextraconstr,liwork,lwork
parameter (liwork=1000,lwork=10000)
integer iwork(liwork)
double precision a(dof,dof),redgradobjf(dof),
$                 hess(dof,dof),obj,ax(n+dof),$
$                 lagrmultnag(dof),work(lwork),lamda(dof),$
$                 Z_basis(n+dof,dof),ub(n+2*dof),lb(n+2*dof),$
$                 x(n),lagrmultng(n+2*dof)$
external E04NFF,E04NFU
nextraconstr=n+dof
ifail=0;istate=0;
call bnd(n,dof,ub,lb,x,lamda)
pz=(ub(n+dof)+lb(n+dof))/2 ! initial guess for the solution of the QP, pz
call E04NFF(dof,nextraconstr,Z_basis,nextraconstr,lb,ub,
$             redgradobjf,hess,dof,
$             E04NFU,istate,pz,iter,objc,lagrmultng,iwork,
$             liwork,work,lwork,ifail)
lagrmultnag(1:dof)=lagrmultng(1:dof)
end

subroutine bnd(n,dof,ub,lb,x,lamda)
implicit none
integer n,dof,i
double precision ub(n+2*dof),lb(n+2*dof),bl_all(n+dof),
$                 bu_all(n+dof),
$                 x(n),lamda(dof),xfull(n+dof)
xfull(l:1:n)=x(l:1:n)
xfull(n+1:n+dof)=lamda(l:1:dof)
call bounds(n,dof,bl_all,bu_all)
do i=1,dof
    ub(i)=bu_all(n+i)-lamda(i)
    lb(i)=bl_all(n+i)-lamda(i)
enddo
do i=dof+1,n+2*dof
    ub(i)=bu_all(i-dof)-xfull(i-dof)
    lb(i)=bl_all(i-dof)-xfull(i-dof)
enddo
end

subroutine reduced_gradient(n,dof,gradobjfun,Z_basis,redgradobjf)
implicit none
integer n,dof
double precision Z_basis(n+dof,dof),Z_basisT(dof,n+dof),
$                 gradobjfun(n+dof),redgradobjf(dof),
$                 temp_all(n+dof,1),temp(dof,1)
Z_basisT=transpose(Z_basis)
temp_all(:,1)=gradobjfun(:)
temp=matmul(Z_basisT,temp_all)
redgradobjf(:)=temp(:,1)
end

subroutine newtonFuZ(n,dof,x,lamda,Z,sizebasis,FuZ)
implicit none
integer i,j,sizebasis,n,dof
double precision,dimension(n)::x,x1,x2,temp1,temp2
double precision lamda(dof),h,Z(n,sizebasis),FuZ(n,sizebasis),
$                 r(sizebasis,sizebasis)
h=1e-6
do j=1,sizebasis
do i=1,n
    x1(i)=x(i)+h*Z(i,j)
    x2(i)=x(i)-h*Z(i,j)
enddo
call G_drv(n,dof,lamda,x1,temp1)
call G_drv(n,dof,lamda,x2,temp2)
do i=1,n
    FuZ(i,j)=(temp1(i)-temp2(i))/(2*h)
enddo
subroutine constraints_gradient(n,dof,x,lamda,constr_grad)
    implicit none
    integer n,dof,i,j
    double precision x(n),lamda(dof),constr_grad(n+dof,n),jac(n,n),$
    & f_lamda(n,dof)
    call jac_drv_full(n,dof,x,lamda,Jac)
    do i=1,n
      do j=1,n
        constr_grad(i,j)=jac(j,i)
      enddo
    enddo ! transpose
    call parameter_sensitivity(n,dof,x,lamda,f_lamda)
    do i=1,n
      do j=1,dof
        constr_grad(n+j,i)=f_lamda(i,j)
      enddo
    enddo
end

subroutine parameter_sensitivity(n,dof,x,lamda,f_lamda)
    implicit none
    integer n,dof
    double precision x(n),lamda(dof),f_lamda(n,dof)
    integer i,j
    double precision lamda_fwd(dof),lamda_bwd(dof),resid_fwd(n),$
    & resid_bwd(n),h,x_fwd(n),x_bwd(n),xnew(n)
    h=1e-6
    do j=1,dof
      lamda_bwd=lamda
      lamda_bwd(j)=lamda(j)-h
      do i=1,n; x_bwd(i)=x(i); enddo
      lamda_fwd=lamda
      lamda_fwd(j)=lamda(j)+h
      do i=1,n; x_fwd(i)=x(i); enddo
      call G_drv(n,dof,lagda_bwd,x_bwd,resid_bwd)
      call G_drv(n,dof,lagda_fwd,x_fwd,resid_fwd)
      do i=1,n; f_lamda(i,j)=(resid_fwd(i)-resid_bwd(i))/(2*h); enddo
    enddo
end

subroutine ssconv0(n,dof,lamda,x,residhistory,maxnewtiter,$
    & sizebasis,tol)
    implicit none
    integer n,maxnewtiter,iter,sizebasis,dof
    double precision x(n),residhistory(n,sizebasis),resid(n),xnew(n),$
    & lagda(dof),tol,lagdanew(dof)
    logical convergence
    lagdanew=lagda
    do iter=1,maxnewtiter
      call newton_step0(n,dof,lagda,x,resid,xnew)
      call conv_criterion(n,dof,x,xnew,lagda,lagdanew,convergence,tol)
      call keep_resid_history(n,iter,sizebasis,resid,residhistory)
      x=xnew
      if (convergence) exit
    enddo
end
Appendix: source code

subroutine newton_step0(n,dof, lamda, x, g, xnew)
implicit none
integer nrhs, n, dof
double precision x(n), g(n), lamda(dof), dz,
\( \mathbf{J} \),
integer ipiv(n), info, lwork
lwork=n
nrhs=1
ipiv=0
call G_drv(n,dof,lamda,x,g)
call jac_drv_full(n,dof,x,lamda,Jac)
call dgesv(n,nrhs,jac,n,ipiv,g,n,info)
xnew=g
if (info.ne.0) then
write(*,*) 'SUBROUTINE DGESV (LAPACK) EXITED WITH INFO=', info
stop 'ifail.ne.0 in dgesv called by newton_step'
endif
xnew=x-xnew
end

subroutine ssconv(n,dof, lamda, x, residhistory, maxnewtiter, sizebasis, tol)
implicit none
integer n, maxnewtiter, iter, sizebasis, dof
double precision x(n), residhistory(n,sizebasis), resid(n), xnew(n),
\( \mathbf{\lambda} \),
logical convergence
do iter=1,maxnewtiter
\call{newton_step}(n,dof, lamda, x, resid, xnew, lamdanew)
call conv_criterion(n, dof, x, xnew, lamda, lamdanew, convergence, tol)
call keep_resid_history(n, iter, sizebasis, resid, residhistory)
x=xnew
lamda=lamdanew
if (convergence) exit
enddo
end

subroutine keep_resid_history(n, iter, sizebasis, resid, residhistory)
implicit none
integer, intent(in):: n, iter, sizebasis
integer i, j
double precision, intent(in):: resid(n)
double precision, dimension(n, sizebasis):: residhistory
do j=sizebasis, 2, -1
do i=1,n
residhistory(i,j)=residhistory(i,j-1)
enddo
residhistory(i,1)=resid(i)
enddo
end

subroutine conv_criterion(n, dof, x, xnew, lamda, lamdanew, convergence, tol)
implicit none
integer, intent(in):: n, dof
integer i
double precision, intent(in) :: x(n), xnew(n), lamda(dof), tol,
   $ lamdanew(dof)

double precision dx(n+dof), norm
logical convergence

do i=1,n
dx(i)=xnew(i)-x(i)
enddo

do i=1,dof
dx(n+i)=lamdanew(i)-lamda(i)
enddo

call normeval((n+dof), dx, norm)
write(*,*) norm

convergence=.false.
if (norm.le.tol) convergence=.true.
end

subroutine initial_guess(n, dof, x, lamda)
imPLICIT none
integer n, dof, i
double precision x(n), lamda(dof)
do i=1,n/2
x(2*(i-1)+1)=0.
x(2*(i-1)+2)=1.
enddo
end

subroutine jac_drv_full(n, dof, x, lamda, Jac)
imPLICIT none
integer n, dof, i, j
double precision, dimension(n) :: x, xl, x2, templ, temp2
double precision jac(n, n), h, lamda(dof)
   h=1e-6

do j=1,n
   do i=1,n
      xl(i)=x(i)
x2(i)=x(i)
   enddo
   xl(j)=xl(j)+h
   x2(j)=x2(j)-h
   call G_drv(n, dof, lamda, xl, templ)
call G_drv(n, dof, lamda, x2, temp2)
do i=1,n
   jac(i, j)=(templ(i)-temp2(i))/(2*h)
enddo
enddo
end

subroutine newton_step(n, dof, lamda, x, g, xnew, lamdanew)
imPLICIT none
integer n, nrhs, lwork, dof, i
parameter(lwork=1000)
double precision x(n), g(n), lamda(dof),
   $ xnew(n), lamdanew(dof), work(lwork),
   $ constr_grad(n+dof), aug_jac(n+n+dof), sol(n+dof)
integer ipiv(n), info
nrhs=1
ipiv=0
call G_drv(n,dof, lamda, x, g)
call constraints_gradient(n,dof, x, lamda, constr_grad)
aug_jac=transpose(constr_grad)

sol(1:n)=g(1:n)
call dgeps('N', n, n+dof, nrhs, aug_jac, n, sol, n+dof, work, lwork, info)
xnew=sol(1:n)
do i=1, dof
    lamdanew(i)=sol(n+i)
enddo
if (info.ne.0) then
    write(*,*) 'SUBROUTINE DGESV (LAPACK) EXITED WITH INFO=', info
    stop 'ifail.ne.0 in dgesv called by newton_step'
endif
xnew=x-xnew
lamdanew=lamda-lamdanew
end

subroutine normeval(dim, array, norm)
implicit none
integer dim, i
double precision norm, array(dim)

norm=0.
do i=1, dim
    norm=norm+array(i)**2
enddo

norm=sqrt(norm)
end

subroutine eye(dm, identity_matrix)
implicit none
integer dm, i, j
double precision identity_matrix(dm, dm)
do i=1, dm
    do j=1, dm
        identity_matrix(i, j)=0.
    enddo
    identity_matrix(i, i)=1.
enddo
end

subroutine calc_Z_basis(n, dof, sizebasis, H, Zh, f_lamda, Z_basis)
implicit none
integer n, dof, sizebasis, i, j, info, ipiv(sizebasis)
double precision H(sizebasis, sizebasis), Zh(n, sizebasis),
$_{\text{f}}$lamda(n, dof), Z_basis(n+dof, dof),
$_{\text{Zext}}$(n+dof, sizebasis+dof), Zr(sizebasis+dof, dof),
$_{\text{Zht}}$(sizebasis, n),
$_{\text{temp1}}$(sizebasis, dof), temp2(n, dof),
$_{\text{temp3}}$(sizebasis, dof), work(sizebasis)

Zext=0.
do i=1, dof
    Zext(n+i, sizebasis+i)=2*lamda
enddo
Zht=transpose(Zh)
A.2 FORTRAN Codes corresponding to Chapter 4

A.2.1 3-dof case study optimised using Algorithm 4.1 (PRSQP)

This program utilizes some of the subroutines of Section A.1.5.

```fortran
subroutine calculate_reduced_Jac(n,dof,x,lamda,Zh,sizebasis,H)
  implicit none
  integer n,dof,sizebasis
  double precision x(n),lamda(dof),Zh(n,sizebasis),FuZ(n,sizebasis),
    $                H(sizebasis,sizebasis),Zht(sizebasis,n)
  call newtonFuZ(n,dof,x,lamda,Zh,sizebasis,FuZ)
  Zht=transpose(Zh)
  H=matmul(Zht,FuZ)
  end
```

program main
  implicit none
  integer n,ni,iter,sizebasis,ncv,maxnewtiter,dof
  double precision sstol
  parameter(n=500,ni=3,sizebasis=10,ncv=20,maxnewtiter=50,dof=3,
    $          sstol=1e-7)
integer i,j,ifail,istate(n+2*dof+ni)
!
!  state variables and parameters

double precision x(n),lamda(dof),s(ni),spr(ni)
!
!  variables for the model reduction procedure

double precision residhistory(n,sizebasis)
!
!  variables for the optimisation procedure

do double precision objfun,gradobjfun(n+dof),pz(dof),a(dof,dof),
$  \text{\texttt{obj}},ax(dof),f_{\text{\texttt{lamda}}}(n,dof),$
$  \text{\texttt{Z}}_{\text{\texttt{basis}}}(n+dof,dof),\text{\texttt{srchdirnorm}},$
$  \text{\texttt{Hess}}(dof,dof),zt(dof,n+dof),$
$  \text{\texttt{bl\_all}}(n+dof),\text{\texttt{bu\_all}}(n+dof),\text{\texttt{lagrmultnag}}(dof),$
$  \text{\texttt{searchdir}}(n+dof),\text{\texttt{redgradobjf}}(dof),\text{\texttt{lag\_mult}}(n),$
$  \text{\texttt{norm}},\text{\texttt{ssx}}(n),zh(n,sizebasis),$
$  \text{\texttt{ep}},\text{\texttt{er}}(sizebasis),\text{\texttt{ei}}(sizebasis),$
$  \text{\texttt{H}}(sizebasis,sizebasis),\text{\texttt{temp1}}(dof,1),\text{\texttt{temp2}},$
$  \text{\texttt{temp}}(n+dof,1),\text{\texttt{pzz}}(dof,1),\text{\texttt{temp3}},\text{\texttt{grd}}(n+dof,1),$
$  \text{\texttt{lag\_mult\_red}}(sizebasis),zh\_prev(n,sizebasis),$
$  \text{\texttt{timeupdZ1}},\text{\texttt{timeupdZ2}},\text{\texttt{ssZ}}(ni,dof),\text{\texttt{lagrms}}(ni)$
logical converged

external E04NHF

!call E04NHF ("Print Level = 0")
!
!residhistory=0.
!lagrmults=0.
!istate=0
!
!open(45,file='optim_path.txt',status='replace')
!write(45,*) 'Iter Parameter_Value'
!write(45,*) '----  ---------------'
!close(45)
!
!open(86,file='br.txt',status='replace');close(86)
!open(86,file='pz.txt',status='replace');close(86)
!open(86,file='redobjgrad.txt',status='replace');close(86)
!open(78,file='lag_multiplier.txt',status='replace'); close(78)
!open(88,file='conv_data.txt',status='replace');close(88)
!open(unit=51,file='timesZupdate.dat',status='replace');close(51)
!
!call initial_guess(n,dof,x,lamda); lamda=2
!
!call eye(dof,Hess)
!
!call bounds(n,dof,bl\_all,bu\_all)
!

! do i=1,10000
!
!  if (i.eq.1) then
!
!    call ssconv0(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
!      sstol)
!
!    ep=1e-3; Zh=residhistory
!    call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
!    call calculate\_reduced\_Jac(n,dof,x,lamda,Zh,sizebasis,H)
!  else
!    call ssconv(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
!      sstol)
!

Appendix: source code

```fortran
ep=5e-2;
call eb22drv2(n_dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
call calculate_reduced_Jac(n_dof,x,lamda,Zh,sizebasis,H)
endif
write(*,*) '***',i,lamda
open(45,file='optim_path.txt',position='append')
write(45,*) 'after newton: lamda=',lamda,x(n),x(n-1), '**end iter',
$ i-1; close(45)
call obj_func(n,x,objfun)
call obj_func_grad(n_dof,x,gradobjfun)
open(87,file='objgrad.txt'); do j=1,n+dof; write(87,*) gradobjfun(j); enddo; close(87)
call parameter_sensitivity(n_dof,x,lamda,f_lamda)
open(87,file='constr.txt'); write(87,*) f_lamda; close(87)
call calc_Z_basis(n_dof,sizebasis,H,Zh,f_lamda,Z_basis)
open(87,file='Z_basis.txt'); do j=1,n+dof;
$ write(87,*) Z_basis(j,dof); enddo; close(87)
if (i.eq.1) then
  ! Zt=transpose(Z_basis); hess=matmul(Zt,Z_basis)
else
  call update_hessian(n_dof,ni,Z_basis,lag_mult,hess,x,lamda,
$    lag_mult_red,lagrmults,sizebasis,Zh_prev)
endif
open(86,file='br.txt',position='append');write(86,*) hess; close(86)
call reduced_gradient(n_dof,gradobjfun,Z_basis,redgradobjf)
open(86,file='redobjgrad.txt',position='append');write(86,*) hess; close(86)
call optimise(n_dof,redgradobjf,hess,pz,lagrmultnag,lamda,
$     Z_basis,x,ni,lagrmults,istate)
open(78,file='current_x.txt'); do j=1,n;write(78,*) x(j);
$   enddo;write(78,*) lamda; close(78)
call update_solution(n_dof,pz,Z_basis,x,lamda,srchdirnorm)
call obj_func(n,x,objfun)
call obj_func_grad(n_dof,x,gradobjfun)
open(87,file='objgrad.txt'); do j=1,n+dof; write(87,*) gradobjfun(j); enddo; close(87)
end!
```

---

```
if (i.eq.1) then
ep=5e-2;
call eb22drv2(n_dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
call calculate_reduced_Jac(n_dof,x,lamda,Zh,Zh_prev=Zh)
endif
```

---

```
call parameter_sensitivity(n_dof,x,lamda,f_lamda)
open(87,file='constr.txt');write(87,*) f_lamda;close(87)
call calc_Z_basis(n_dof,sizebasis,H,Zh,f_lamda,Z_basis)
open(87,file='Z_basis.txt'); do j=1,n+dof;
```
Appendix: source code

$       write(87,*) Z_basis(j,dof); enddo; close(87)

Zt=transpose(Z_basis);grd(1:n+dof,1)=gradobjfun(1:n+dof);
    temp=matmul(Zt,grd);
call normeval(dof,temp,templ,temp3)
pzz(1:dof,1)=pz(1:dof);temp=matmul(Z_basis,pzz);
call normeval(n+dof,temp,temp2)
open(88,file='conv_data.txt',position='append')
write(88,*) i,lamda,objfun,temp3,temp2
close(88)

Zh_prev=Zh
call lagrange_multiplier(n,dof,gradobjfun,lag_mult,sizebasis,Zh,H,
    lag_mult_red,ni,lagrmults,x,lamda)
open(78,file='lag_multipliers.txt');do j=1,n;
$      write(78,*) lag_mult(j); enddo; close(78)
write(*,*) '*** main iteration:',i,'lamda=',lamda,'***'
call normeval(n+dof,matmul(Z_basis,pz),norm)
open(45,file='optim_path.txt',position='append')
write(45,*) i,lamda,x(n),x(n-1)
write(45,*) objfun,matmul(transpose(Z_basis),gradobjfun),norm
close(45)
open(45,file='optim_data1.txt',position='append')
write(45,*) i,lamda
open(45,file='optim_data2.txt',position='append')
write(45,*) i, objfun;
close(45)
open(45,file='optim_data3.txt',position='append')
write(45,*) i, matmul(transpose(Z_basis),gradobjfun);
close(45)
open(45,file='optim_data4.txt',position='append')
write(45,*) i, norm;
close(45)
call sqp_conv_criterion(srchdirnorm,converged)
write(*,*) i,lamda,x(n-1),x(n)
if ((converged).and.(i.gt.1)) exit
open(86,file='nonlin_behav.txt',position='append')
    write(86,*) i, lagrmults; close(86)
enddo
write(*,*) 'Reduced Hessian method converged. lamda=',lamda
end ! of main program

subroutine eval_ineq_f(n,ni,x,ss)
    implicit none
    integer n,ni,i,k
    double precision x(n),ss(ni)
k=0
do i=n/2,n/2+(ni-1)*4,4 !n/2+2*(ni-1),2
    k=k+1
!       the nonlinear ineqs are x^2<7 for a region in the centre of the
    ss(k)=160-(x(i)+x(i+2))**2 !!!7-x(i)**2
    enddo
end
subroutine aineq(n,dof,ni,x,lambda,alhs,alb)
! computes the inequalities for the qp subproblem
! c3->inequalities, cx3->derivative of the inequalities wrt dependent
! variables
implicit none
integer n,dof,ni
double precision x(n),lambda(dof),z_basis(n+dof,dof),c3(ni),
$ cx3(ni),alhs(ni,dof),alb(ni),tZ(n,dof),
$ cx3T(ni,n)
call eval_ineq_f(n,ni,x,c3)
call perturb_inequalities(n,dof,ni,x,lambda,cx3)
tZ(1:n,1:dof)=z_basis(1:n,1:dof)
cx3T=transpose(cx3)
alhs=matmul(cx3T,tZ)
alb=-c3
end

subroutine perturb_inequalities(n,dof,ni,x,lambda,cx3)
!!! subroutine to compute cx3=derivative of ineq.constr wrt. x. formally
ssZ
implicit none
integer n,dof,ni,i,j,z_2nddim,z_1stdim
double precision h,x(n),xf(n),xb(n),ssf(ni),ssb(ni),
$ lambda(dof),lambdaf(dof),
$ lambdab(dof),t(ni),cx3(n,ni)
h=1e-6
do j=1,n !!!z_2nddim
  xf=x+h !!!*z_basis(1:n,j)
xh=x-h !!!*z_basis(1:n,j)
  lambdaf=lambda!+z_basis(n+1:n+dof,j)
lambdab=lambda!-z_basis(n+1:n+dof,j)
! ATTENTION:
! in the general case, lambda needs to be perturbed as well. However, here,
! inequality constraints are only a function of the dependent variables.
call eval_ineq_f(n,ni,xf,ssf)
call eval_ineq_f(n,ni,xb,ssb)
t=(ssf-ssb)/(2*h)
do i=1,ni
  ssZ(i,j)=t(i)
cx3(j,i)=t(i)
enddo
dendo
dend

subroutine G_drv(n,dof,lambda,x,g)
integer n,dof
double precision x(n),lambda(dof),g(n)
call func(n,x,lambda,g)
end

subroutine func(n,x,par,g)
! equivalent to: g_drv(n,dof,lambda,x,g)
implicit none
integer nn,n,i,nact,dof,k
parameter (nact = 3,dof=3)
double precision x(n),x1(n/2),x2(n/2),g1(n/2),g2(n/2),
  dx,g(n),par(dof)
double precision pe1,pe2,da,gamma,b,le,beta,x2w
double precision uact(nact)
integer switch(nact),fl1,fl2,fl3
dx = 1.d0 / dble((n/2) - 1)
nn = n / 2
k=0
do i = 1,nn
  k=k+1
  x1(i) = x(k)
  k=k+1
  x2(i) = x(k)
end do

* set parameters values
b = 12.
beta = 1.5d0
pe1 = 5.
pe2 = 5.d0
le = 1.
gamma = 20.
da = 0.1d0
do i=1, nact
  uact(i) = par(i)
endo
switch(1) = 1
switch(2) = 0
switch(3) = 0
call u_function(n, nact, switch, uact, x2w)
g1(1) = (2.*(x1(2)-(dx*pe1+1.)*x1(1)))/(pe1*dx*dx) - pe1*x1(1)
  + da*(1.-x1(1))*dexp(x2(1)/(1.+(x2(1)/gamma)))
g2(1) = (2.*(x2(2)-(dx*pe2+1.)*x2(1)))/(le*pe2*dx*dx)
  - pe2*x2(1)/le - beta*x2(1)/le
  + (b*da/le)*(1.-x1(1))*dexp(x2(1)/(1.+(x2(1)/gamma)))
  + beta*x2w/le
endo
z = dble(i - 1)/dble(nn-1)
if (z.le.1.d0/3.d0) then
  switch (1) = 1
  switch (2) = 0
  switch (3) = 0
  call u_function(n, nact, switch, uact, x2w)
endif
if ((z.gt.1.d0/3.d0).and.(z.le.2.d0/3.d0)) then
  switch (1) = 0
  switch (2) = 1
  switch (3) = 0
  call u_function(n, nact, switch, uact, x2w)
end if
if (z.gt.2.d0/3.d0) then
  switch(1) = 0
  switch(2) = 0
  switch(3) = 1
  call u_function(n, nact, switch, uact, x2w)
enddo
switch = 1
switch = 0
switch = 1
call u_function(n, nact, switch, uact, x2w)
g1(nn) = 2.*(x1(nn-1)-x1(nn))/(pe1*dx*dx)
  + da*(1.-x1(nn))*dexp(x2(nn)/(1.+(x2(nn)/gamma)))
g2(nn) = 2.*(x2(nn-1)-x2(nn))/(le*pe2*dx*dx)
  - beta*x2(nn)/le
  + (b*da/le)*(1.-x1(nn))*dexp(x2(nn)/(1.+(x2(nn)/gamma)))
  + beta*x2w/le


```fortran
k=0
do i=1,n/2
  k=k+1
  g(k)=g1(i)
  k=k+1
  g(k)=g2(i)
enddo
end

subroutine u_function(n,nact,switch,uact,u)
  implicit none
  integer n,nact,j,switch(nact)
  double precision u, UAct(nact)
  u=0.d0
  do j=1,nact
    u = u + dble(switch(j)) * uact(j)
  end do
end

subroutine lagrange_multiplier(n,dof,gradobjfun,lag_mult,
                               sizebasis,Zh,H,lag_mult_red,
                               ni,lagrmults,x,nrhs)
  implicit none
  integer n,dof,ipiv(n),info,i,j,k,nrhs,sizebasis,ni
  parameter(nrhs=1)
  double precision x(n),lamda(dof),gradobjfun(n+dof),
  lag_mult(n),temp_n(n,nrhs),temp_all(n+dof,nrhs),
  work(n),Zh(n,sizebasis),lag_mult_redl(sizebasis),
  H(sizebasis,sizebasis),temp(sizebasis,nrhs),
  ZhT(sizebasis,n),invH(sizebasis,sizebasis),
  temp_backproj(n,nrhs),temp_graddof(n,nrhs),
  lagrmults(n),lagrmults1(ni,nrhs),
  norm,
  ZhTcx3(sizebasis,n),lagrmults1(ni,nrhs),
  temp1(sizebasis,nrhs)

  ZhT=transpose(Zh)
  lwork=n
  invH=transpose(H)
  do i=1,n
    temp_graddof(i,1)=gradobjfun(i)
  enddo
  temp=matmul(ZhT,temp_graddof)

! the extra bit for the inequalities--------------------------------------
call perturb_inequalities(n,dof,ni,x,lamda,cx3)
  ZhTcx3=matmul(ZhT,cx3)
  lagrmults1(:,nrhs)=lagrmults(:)
  temp=matmul(ZhTcx3,lagrmults1)
  temp_backproj=matmul(Zh,temp)
  temp=-temp
call dgesv(sizebasis,nrhs,invH,sizebasis,ipiv,temp,sizebasis,info)
if (info.ne.0) stop 'error from dgesv in lagr.multipl. calculation'
  temp_backproj=matmul(Zh,temp)
```
do i=1,sizebasis
lag_mult_red(i)=temp(i,1)
enddo
doi=1,n
lag_mult(i)=temp_backproj(i,1)
enddo

open(78,file='lag_multipliers.txt',position='append')
doi=1,n;write(78,*)lag_mult(i);enddowrite(78,*)'----------'
close(78)
end

subroutine optimise(n,dof,redgradobjf,hess,pz,lagrmultnag, $lamda,Z_basis,x,ni,lagrmults,istate)
imPLICIT none
integer n,ni,dof,ifail,istate(n+2*dof+ni),iter,nextraconstr, $lwork,liwork,k,i
PARAMETER (liwork=1000,lwork=10000)
ninteger iwork(liwork)
double precision a(n+dof+ni,dof),redgradobjf(dof), $hess(dof,dof),pz(dof),obj,ax(n+dof+ni), $lagrmultnag(dof),work(lwork),lamda(dof), $Z_basis(n+dof,dof),ub(n+2*dof+ni), $lb(n+2*dof+ni),x(n),lagrmultng(n+2*dof+ni), $ssZ(ni,dof),lagrmslts(ni),alhs(ni,dof),alb(ni)
eXTERNAL E04NFF,E04NFW
CALL alineq(n,dof,ni,x,lamda,alhs,alb)
nextraconstr=n+dof+ni
A(1:n+dof,1:dof)=Z_basis(:,,:)
a(n+dof+1:n+dof+ni,1:dof)=alhs(:,:)
ifail=1;istate=0;
call bnd(n,dof,ni,ub,lb,x,lamda,alb)
pz=(ub(n+dof)+lb(n+dof))/2 ! initial guess for the solution of the QP, pz
  CALL E04NFF(dof,nextraconstr,a,nextraconstr,lb,ub, $redgradobjf,hess,dof, $hess(dof,dof),pz(dof),obj,ax(n+dof+ni), $lagrmultnag(dof),work(lwork),lamda(dof), $Z_basis(n+dof,dof),ub(n+2*dof+ni), $lb(n+2*dof+ni),x(n),lagrmultng(n+2*dof+ni), $ssZ(ni,dof),lagrmults(ni),alhs(ni,dof),alb(ni)
  CALL E04NFWU,istate,pz,iter,obj,ax,lagrmultng,iwork, $liwork,work,lwork,ifail)
lagrmultnag(1:dof)=lagrmultng(1:dof)
doi=1,ni
lagrmults(i)=lagrmultng(n+2*dof+1)
enddo
end

subroutine obj_func_grad(n,dof,x,gradobjfun)
imPLICIT none
integer n,dof
double precision x(n),gradobjfun(n+dof)
graddobjfun=0.
graddobjfun(n-1)=-1
end

subroutine obj_func(n,x,objfun)
imPLICIT none
integer n
double precision x(n),objfun
objfun=-x(n-1)
end
subroutine bounds(n,dof,bl_all,bu_all)
  implicit none
  integer n,dof,i
  double precision bl_all(n+dof),bu_all(n+dof)
  do i=1,n/2
    bl_all(2*(i-1)+1)=0
    bl_all(2*(i-1)+2)=0
    bu_all(2*(i-1)+1)=1
    bu_all(2*(i-1)+2)=8
  enddo
  do i=1,dof
    bl_all(n+i)=0
    bu_all(n+i)=4
  enddo
end

subroutine bnd(n,dof,ni,ub,lb,x,lamda,alb)
  implicit none
  integer n,dof,i,ni
  double precision ub(n+2*dof+ni),lb(n+2*dof+ni),bl_all(n+dof),
  & bu_all(n+dof),
  & x(n),lamda(dof),xfull(n+dof),alb(ni)
  xfull(1:n)=x(1:n)
  xfull(n+1:n+dof)=lamda(1:dof)
  call bounds(n,dof,bl_all,bu_all)
  do i=1,dof
    ub(i)=bu_all(n+i)-lamda(i)
    lb(i)=bl_all(n+i)-lamda(i)
  enddo
  do i=dof+1,n+2*dof
    ub(i)=bu_all(i-dof)-xfull(i-dof)
    lb(i)=bl_all(i-dof)-xfull(i-dof)
  enddo
  do i=n+2*dof+1,n+2*dof+ni
    ub(i)=1e+36
  enddo
  lb(n+2*dof+1:n+2*dof+ni)=alb
end

subroutine update_hessian(n,dof,ni,Z_basis,lag_mult,hess,x,lamda,
  & lag_mult_red,lagrmults,sizebasis,Zh)
  implicit none
  integer n,dof,i,j,k,one,zero,sizebasis,ni
  parameter(one=1,zero=0)
  double precision x(n),Z_basis(n+dof,dof),hess(dof,dof),lag_mult(n)
  & xfull(n+dof),lamda(dof),hfullhess(dof,dof,ni),sstol,
  & objfun1,objfun2,objfun3,objfun4,hessobjf(dof,dof)
  & residhistory(n,one),Zh(n,sizebasis),
  & temp_n(n,one),lagrmults(ni),fullhess_ni(dof,dof,ni)
  double precision,dimension(n+dof):xfull1,xfull2,xfull3,xfull4
  double precision,dimension(n):f1,f2,f3,f4
  h=1e-3
  sstol=1e-7
  xfull(1:n)=x(1:n); xfull(n+1:n+dof)=lamda(1:dof)
  ZhT=transpose(Zh)
  do i=1,dof
do j=1,dof
  do k=1,n+dof
    xfull1(k)=xfull(k)+h*Z_basis(k,i)+h*Z_basis(k,j)
    xfull2(k)=xfull(k)+h*Z_basis(k,i)-h*Z_basis(k,j)
    xfull3(k)=xfull(k)-h*Z_basis(k,i)+h*Z_basis(k,j)
    xfull4(k)=xfull(k)-h*Z_basis(k,i)-h*Z_basis(k,j)
  enddo
  call G_drv(n,dof,xfull1(n+1:n+dof),xfull1(1:n),f1)
  call G_drv(n,dof,xfull2(n+1:n+dof),xfull2(1:n),f2)
  call G_drv(n,dof,xfull3(n+1:n+dof),xfull3(1:n),f3)
  call G_drv(n,dof,xfull4(n+1:n+dof),xfull4(1:n),f4)
  do k=1,n
    fullhess(i,j,k)=(f1(k)-f2(k)-f3(k)+f4(k))/((2*h)**2)
  enddo
  call eval_ineq_f(n,ni,xfull1(1:n),f1(1:ni)) !f1(1:ni) is reused for ineq
  call eval_ineq_f(n,ni,xfull2(1:n),f2(1:ni)) !f2(1:ni) is reused for ineq
  call eval_ineq_f(n,ni,xfull3(1:n),f3(1:ni)) !f3(1:ni) is reused for ineq
  call eval_ineq_f(n,ni,xfull4(1:n),f4(1:ni)) !f4(1:ni) is reused for ineq
  ! ATTENTION: reusing fi only holds for N=ni, hence dim(G)<=dim(h)
  do k=1,ni
    fullhess_ni(i,j,k)=(f1(k)-f2(k)-f3(k)+f4(k))/((2*h)**2)
  enddo
  call obj_func(n,xfull1(1:n),objfun1)
  call obj_func(n,xfull2(1:n),objfun2)
  call obj_func(n,xfull3(1:n),objfun3)
  call obj_func(n,xfull4(1:n),objfun4)
  hessobjf(i,j)=(objfun1-objfun2-objfun3+objfun4)/((2*h)**2)
enddo
enddo

A.2.2 3-dof case study optimised using Algorithm 4.2 (KS)

This program utilizes the procedures of Section A.1.5.

program main
implicit none
integer n,iter,sizebasis,ncv,maxnewtiter,dof
double precision sstol
parameter(n=500,sizebasis=10,ncv=20,maxnewtiter=50,dof=3,
$sstol=1e-7)$

...
integer i,j,ifail
!
state variables and parameters

double precision x(n),lamda(dof)
!
variables for the model reduction procedure

double precision residhistory(n,sizebasis)
!
variables for the optimisation procedure

double precision objfun,gradobjfun(n+dof),a(dof,dof),
$\quad$ istate(dof),obj,ax(dof),f_lamda(n,dof),
$\quad$ Z_basis(n+dof,dof),srchdirnorm,
$\quad$ Hess(dof,dof),Zt(dof,n+dof),
$\quad$ bl_all(n+dof),bu_all(n+dof),lagrmultnag(dof),
$\quad$ searchdir(n+dof),redgradobjf(dof),lag_mult(n),
$\quad$ norm,ssx(n),Zh(n,sizebasis),
$\quad$ ep,er(sizebasis),ei(sizebasis),
$\quad$ H(sizebasis,sizebasis),temp1(dof,1),temp2,
$\quad$ temp(n+dof,1),pzz (dof,1),temp3,grd(n+dof,1),
$\quad$ lag_mult_red(sizebasis),Zh_prev(n,sizebasis)

logical converged

residhistory=0.

open(45,file='optim_path.txt',status='replace')
write(45,*) 'Iter  Parameter_Value'
write(45,*) '----  ---------------'
close(45)

open(86,file='br.txt',status='replace');close(86)
open(86,file='pz.txt',status='replace');close(86)
open(86,file='redobjgrad.txt',status='replace');close(86)
open(78,file='lag_multipliers.txt',status='replace'); close(78)
open(88,file='conv_data.txt',status='replace');close(88)
open(47,file='linesearchbehaviour.txt',status='replace');close(47)

call initial_guess(n,dof,x,lamda); lamda=2

call eye(dof,Hess)

call bounds(n,dof,bl_all,bu_all)

do i=1,100

if (i.eq.1) then
  call ssconv0(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
$\quad$ sstol)
  ep=1e-3; Zh=residhistory
  call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
  call calculate_reduced_Jac(n,dof,x,lamda,Zh,sizebasis,H)
else
  call ssconv(n,dof,lamda,x,residhistory,maxnewtiter,sizebasis,
$\quad$ sstol)
  ep=5e-2;
  call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lamda,er,ei)
  call calculate_reduced_Jac(n,dof,x,lamda,Zh,sizebasis,H)
endif
write(*,*) '***',i,lamda
open(45,file='optim_path.txt',position='append')
write(45,*) 'after newton: lamda=',lamda,x(n),x(n-1),'**end iter',
$\quad$ i-1; close(45)
Appendix: source code

call obj_func(n,x,objfun)
call obj_func_grad(n,dof,x,gradobjfun)
open(87,file='objgrad.txt');do j=1,n+dof; write(87,*) gradobjfun(j); enddo; close(87)
call parameter_sensitivity(n,dof,x,lambda,f_lambda)
open(87,file='constr.txt');write(87,*) f_lambda;close(87)
call calc_Z_basis(n,dof,sizebasis,H,Zh,f_lambda,Z_basis)
open(87,file='Z_basis.txt');do j=1,n+dof;
$   write(87,*) Z_basis(j,dof); enddo; close(87)
if (i.eq.1) then !
   Zt=transpose(Z_basis); hess=matmul(Zt,Z_basis)
else
call update_hessian(n,dof,Z_basis,lag_mult,hess,x,lambda,
$    lag_mult_red,sizebasis,Zh_prev)
endif
open(86,file='br.txt',position='append');write(86,*) hess;
close(86)
call reduced_gradient(n,dof,gradobjfun,Z_basis,redgradobjf)
open(86,file='redobjgrad.txt',position='append');write(86,*)
$   hess;close(86)
call optimise(n,dof,redgradobjf,hess,pz,lagrmultnag,lambda,
$   Z_basis,x)
open(78,file='current_x.txt');do j=1,n;write(78,*) x(j);
$   enddo;write(78,*) lamda;close(78)
call update_solution(n,dof,pz,Z_basis,x,lambda,srchdirnorm)
call obj_func(n,x,objfun)
call obj_func_grad(n,dof,x,gradobjfun)
open(87,file='objgrad.txt');do j=1,n+dof; write(87,*)
$   gradobjfun(j); enddo; close(87)
ep=5e-2;
call eb22drv2(n,dof,x,ep,Zh,ncv,sizebasis,lambda,er,ei)
call calculate_reduced_Jac(n,dof,x,lambda,Zh,sizebasis,H)
Zh_prev=Zh
!-------------------
call parameter_sensitivity(n,dof,x,lambda,f_lambda)
open(87,file='constr.txt');write(87,*) f_lambda;close(87)
call calc_Z_basis(n,dof,sizebasis,H,Zh,f_lambda,Z_basis)
open(87,file='Z_basis.txt');do j=1,n+dof;
$   write(87,*) Z_basis(j,dof); enddo; close(87)
Zt=transpose(Z_basis);grd(1:n+dof,1)=gradobjfun(1:n+dof);
    temp1=matmul(Zt,grd);
call normeval(dof,temp1,temp3)
pzz(1:dof,1)=pz(1:dof);temp=matmul(Z_basis,pzz);
call normeval(n+dof,temp,temp2)
open(88,file='conv_data.txt',position='append')
write(88,*) i,lambda,objfun,temp3,temp2
close(88)
Zh_prev=Zh
call lagrange_multiplier(n,dof,gradobjfun,lag_mult,sizebasis,Zh,H,
$                         lag_mult_red)
open(78,file='lag_multipliers.txt');do j=1,n;
$      write(78,*) lag_mult(j); enddo; close(78)
write(*,*) '*** main iteration:',i,'lamda=',lamda,'***'
call normeval(n+dof,matmul(Z_basis,pz),norm)
open(45,file='optim_path.txt',position='append')
write(45,*) i,lamda,x(n),x(n-1)
write(45,*) objfun,matmul(transpose(Z_basis),gradobjfun),norm
close(45)
call sqp_conv_criterion(srchdirnorm,converged)
write(*,*) i,lamda,x(n-1),x(n)
if ((converged).and.(i.gt.1)) exit
enddo
write(*,*) 'Reduced Hessian method converged. lamda)'),lamda
end ! of main program

subroutine G_drv(n,dof,lamda,x,g)
   implicit none
   integer n,dof
   double precision x(n),lamda(dof),g(n)
call func(n,x,par,g)
end

subroutine func(n,x,par,g)  ! equivalent to: g_drv(n,dof,par,par)
   implicit none
   integer nn,n,i,nact, dof,k
   parameter (nact = 3,dof=3)
   double precision x(n),x1(n/2),x2(n/2),g1(n/2),g2(n/2),
*                 dx,g(n),par(dof)
   double precision pe1,pe2,da,gamma,b,le,beta,x2w
   double precision z,uact(nact)
   integer switch(nact),fl1,fl2,fl3
   dx = 1.d0 / dble((n/2) - 1)
nn = n / 2
   k=0
   do i = 1,nn
      k=k+1
      x1(i) = x(k)
      k=k+1
      x2(i) = x(k)
   end do
*     set parameters values
   b=  12.
   beta=  1.5d0
   pe1=  5.
   pe2=  5.d0
   le=  1.
   gamma=  20.
   da = 0.1d0
   do i=1, nact
Appendix: source code

```
!source code

uact(i) = par(i)
enddo
switch(1) =1
switch(2)= 0
switch(3) =0
call u_function(n,nact,switch,uact,x2w)
g1(1)=(2.**(x1(2)-(dx*pe1+1.)*x1(1)))/(pe1*dx*dx) - pe1*x1(1)
1 + da*(1.-x1(1))*dexp(x2(1)/(1.+x2(1)/gamma)))
g2(1)=(2.**(x2(2)-(dx*pe2+1.)*x2(1)))/(le*pe2*dx*dx)
1 - pe2*x2(1)/le - beta*x2(1)/le
2 + (b*da/le)*(1.-x1(1))*dexp(x2(1)/(1.+x2(1)/gamma)))
3 + beta*x2w/le
do i = 2,nn-1
  z = dble(i - 1)/dble(nn-1)
  if (z.le.1.d0/3.d0) then
    switch (1) = 1
    switch (2) = 0
    switch (3) = 0
    call u_function(n,nact,switch,uact,x2w)
  endif
  if ((z.gt.1.d0/3.d0).and.(z.le.2.d0/3.d0)) then
    switch (1) = 0
    switch (2) = 1
    switch (3) = 0
    call u_function(n,nact,switch,uact,x2w)
  end if
  if (z.gt.2.d0/3.d0) then
    switch(1)=0
    switch(2)=0
    switch(3)=1
    call u_function(n,nact,switch,uact,x2w)
  endif
  g1(i)=(x1(i+1)-2.*x1(i)+x1(i-1))/(pe1*dx*dx)
1 - (x1(i+1)-x1(i-1))/(2.*dx)
2 + da*(1.-x1(i))*dexp(x2(i)/(1.+x2(i)/gamma)))
g2(i)=(x2(i-1)-2.*x2(i)+x2(i+1))/(le*pe2*dx*dx)
1 - (x2(i+1)-x2(i-1))/(2.*le*dx) - beta*x2(i)/le
2 + (b*da/le)*(1.-x1(i))*dexp(x2(i)/(1.+x2(i)/gamma)))
3 + beta*x2w/le
endo
switch(1) = 0
switch(2) = 0
switch(3) = 1
call u_function(n,nact,switch,uact,x2w)
g1(nn)=2.*(x1(nn-1)-x1(nn))/(pe1*dx*dx)
1 + da*(1.-x1(nn))*dexp(x2(nn)/(1.+x2(nn)/gamma)))
g2(nn)=2.*(x2(nn-1)-x2(nn))/(le*pe2*dx*dx) - beta*x2(nn)/le
1 + (b*da/le)*(1.-x1(nn))*dexp(x2(nn)/(1.+x2(nn)/gamma)))
2 + beta*x2w/le
k=0
do i=1,n/2
  k=k+1
  g(k)=g1(i)
  k=k+1
  g(k)=g2(i)
enddo
end

subroutine u_function(n,nact,switch,uact,u)
implicit none
integer n,nact,j,switch(nact)
double precision u, UACt(nact)
u=0.d0
do j=1,nact
  u = u + dble(switch(j)) * uact(j)
enddo
```
subroutine bounds(n,dof,bl_all,bu_all)
  implicit none
  integer n,dof,i
  double precision bl_all(n+dof),bu_all(n+dof)
  do i=1,n/2
    bl_all(2*(i-1)+1)=0
    bl_all(2*(i-1)+2)=0
    bu_all(2*(i-1)+1)=1
    bu_all(2*(i-1)+2)=8
  enddo
  do i=1,dof
    bl_all(n+i)=0
    bu_all(n+i)=4.
  enddo
end

subroutine obj_func_grad(n,dof,x,gradobjfun)
  implicit none
  integer n,dof,i
  double precision x(n),gradobjfun(n+dof),objfunfwd,objfunbwd,$
                 xfwd(n),xbwd(n),eps
  gradobjfun=0.
  eps=1e-5
  do i=1,n
    xfwd=x
    xfwd(i)=x(i)+eps
    xbwd=x
    xbwd(i)=x(i)-eps
    call obj_func(n,xfwd,objfunfwd)
    call obj_func(n,xbwd,objfunbwd)
    gradobjfun(i)=(objfunfwd-objfunbwd)/(2*eps)
  enddo
end

subroutine obj_func(n,x,objfun)
  implicit none
  integer n,i
  double precision x(n),objfun,rho,sum1,sum2,small,M,xx(n),ub
  parameter(rho=1d+1)
  ub=7
  xx=x/ub
  small=1e-300
  objfun=-xx(n-1)
  sum1=0
  sum2=0
  M=0
  do i=250,260,4
    sum1=sum1+exp(rho*(xx(i)**2+xx(i+2)**2-2*(6.0/ub)**2-2*M))
  enddo
  objfun=objfun+(1/rho)*log(sum1)+2*M
  open(51,file='obj_data.dat',position='append')
  write(51,'(E15.5,1X,E15.5,1X,E15.5,1X,E15.5,1X,E15.5)')
              M,sum1,objfun,x(250),x(250)**2-2.5**2
  close(51)
end
A.3 MATLAB Codes corresponding to Chapter 5

A.3.1 Implementation of the control Algorithm 5.2

% Clear workspace for the preparation of the execution
clear all; close all;

% Specify system dimensions
n_in = 8;  %%size(B,2);  %# inputs
n_out = 1;  %%size(C,1);  %# outputs
n_states = 32;  %%size(A,1);  %# states
nb=4;  % size of the basis for the dominant subspace

% Specify horizons
M = 7;  %Control Horizon
N = M;  %Prediction Horizon

% Specify weights of the MPC
r = diag(ones(1,n_in));  %Weights on input moves (i.e. ||u(t)-u(t-1)||)
q = diag(4*ones(1,n_out));  %Weights on output deviation from setpoint

% Plant constraints
lb = zeros(n_in,1);  ub = ones(n_in,1);

% Simulation parameters
sim_length = 10;  %Simulation length in sec
T = 100*sim_length+1;  %sim_length/Ts; %number of samples in simulation
Ts=sim_length/(T-1);
t = 0:Ts:sim_length;  %time steps

% Initial Matrix Filling (to reduce the computational cost)
Lamda = zeros((N+1)*n_out,nb);
Z=zeros(n_states,nb);

% Computation of matrices which can be calculated offline
Q = sparse(kron(eye(N+1),q));
R = kron(diag([2*ones(1,N),1]),r);
R = R + kron(diag(-ones(1,N),-1),r);
R = R + kron(diag(-ones(1,N),1),r);
Lb = kron(ones((M+1),1),lb);  Ub = kron(ones((M+1),1),ub);

% Set system initial conditions......
x = zeros(n_states,T);  % x is the incremental variable: xx-ref
xx = zeros(n_states,T);  % xx is the full states vector
xx(:,1) = zeros(n_states,1);  % Plant state
xh = zeros(n_states+n_out,T);  % Estimated state
u = zeros(n_in,T);  % Input record
u_old = zeros(n_in,1);
y = zeros(n_out,T);  % Output record
zxx = zeros(nb,T);  % reduced full states vector
zxx(:,1) = zeros(nb,1);  % reduced Plant state
zxh = zeros(nb+n_out,T);  % reduced Estimated state

% Generate reference signal
[ref]=generate_reference(sim_length,xx(:,1),u(:,1));

tm=cputime();
tic
opt=optimset('display','off');

% MPC loop: Simulation of the closed loop system
for i = 1:T-1,
%%% simulate real plant using the time integrator
xx(:,i+1)=time_integration(Ts,xx(:,i),u(:,i));
x(:,i+1)=xx(:,i+1)-xx(:,i); %refx(:,i+1);

%%% A,B,C,D come from gradient information and are discretized in time
[A,B,C,D,Z]=linearise_nonlinear_cstr(Ts,xx(:,i),u(:,i),nb);

zxx(:,i)=Z'*xx(:,i);
y(:,i) = C*zxx(:,i) + D*u(:,i);  % should be a function of xx, not x

% Matrix fill
[Lambda Phi]=largematrices(N,M,n_in,n_out,nb,A,B,C,D);
H = full(Phi'*Q*Phi) + R;
fd = full([Phi'*Q*Lamda, Phi'*kron(ones(N+1,1),q)]);

% Add disturbance
if i>300, y(:,i)=y(:,i)+0.005*ones(n_out,1); end

%CONTROLLER STARTS HERE-----------------------------

%Observe state
xh(:,i+1) =[x(:,i+1); y(:,i)];
zxh(:,i+1) =[Z'*x(:,i+1); y(:,i)];

% Solve dynamic optimisation problem
f = fd*[Z'*xh(1:n_states,i+1);xh(n_states+1:end,i+1)-ref(:,i)];
f(1:n_in) = f(1:n_in)-r*u_old;

U = quadprog(H,f,[],[],[],[],Lb,Ub,[],opt);
u(:,i+1) = U(1:n_in); u_old = u(:,i);

end;

tm=cputime()-tm;
fprintf(1,'\n \t Closed loop simulation took %f sec\n',tm);
toc

% Plots
figure;
plot(t(1:end-1),ref(1:end-1),'r--','linewidth',1.5); hold on
plot(t(1:end-1),y(1:end-1),'k-','linewidth',1); hold off;
ylabel('Output and reference');
xlabel('Time (dimensionless)');

figure;
plot([0 sim_length],[ub ub],'r--'); hold on;
plot([0 sim_length],[lb lb],'r--');
stairs(t(1:end-1),u(:,1:end-1),'b-'); hold off;
ylabel('Time (dimensionless)');
ylabel('Input and constraints');

A.3.2 Implementation of the control Algorithm 5.3

% Clear workspace for the preparation of the execution
clear all; close all;

% update the basis every <max_iter_without_update> at most
max_iter_without_update=50;

% Specify system dimensions
n_in = 8;  %%size(B,2);    # inputs
n_out = 1;  %%size(C,1);   # outputs
n_states = 32;  %%size(A,1);  # states
nb=4;  % size of the basis for the dominant subspace

% Specify horizons
M = 7;  %Control Horizon
N = M;  %Prediction Horizon
% Specify weights of the MPC
r = diag(ones(1,n_in));  %Weights on input moves (i.e. ||u(t)-u(t-1)||)
q = diag(4*ones(1,n_out));  %Weights on output deviation from setpoint

% Plant constraints
lb = zeros(n_in,1); ub = ones(n_in,1);

% Simulation parameters
sim_length = 10;  %Simulation length in sec
T = 100*sim_length+1;  %number of samples in simulation
Ts = sim_length/(T-1);
t = 0:Ts:sim_length;  %time steps

% Initial Matrix Filling (to reduce the computational cost)
Lamda = zeros((N+1)*n_out,nb);
Phi = zeros((N+1)*n_out);
AB = zeros((N+1)*n_out,n_in);
Z=zeros(n_states,nb);

% Computation of matrices which can be calculated offline
Q = sparse(kron(eye(N+1),q));
R = kron(diag([2*ones(1,N),1]),r);
R = R + kron(diag(-ones(1,N),-1),r);
R = R + kron(diag(-ones(1,N),1),r);
Lb = kron(ones((M+1),1),lb); Ub = kron(ones((M+1),1),ub);

% Set system initial conditions......
x = zeros(n_states,T);        % x is the incremental variable: xx-ref
xx = zeros(n_states,T);       % xx is the full states vector
xx(:,1) = zeros(n_states,1);  % Plant state
xh = zeros(n_states+n_out,T); % Estimated state
u = zeros(n_in,T);            % Input record
u_old = zeros(n_in,1);
y = zeros(n_out,T);           % Output record
zxx = zeros(nb,T);            % reduced full states vector
zxx(:,1) = zeros(nb,1);       % reduced Plant state
zxh = zeros(nb+n_out,T);      % reduced Estimated state

% Generate reference signal
[ref]=generate_reference(sim_length,xx(:,1),u(:,1));

% MPC loop: Simulation of the closed loop system
for i = 1:T-1,
    %% simulate real plant using the time integrator
    xx(:,i+1)=time_integration(Ts,xx(:,i),u(:,i));
x(:,i+1)=xx(:,i+1)-xx(:,i);
    if (i>10) && (norm(xx(:,i+1)-xx(:,i))/norm(xx(:,i+1)))>0.01)
        fprintf(1,'Z updated
')
end

% Matrix fill
[Lamda Phi]=largematrices(N,M,n_in,n_out,nb,A,B,C,D);
W = full(Phi'*Q*Phi) + R;
f = full((Phi'*Q*Lamda, Phi'*kron(ones(N+1,1),q)));
fprintf(1,'Z updated
')
else
    fprintf(1,'no_update\n')
end
zxx(:,i)=Z'*xx(:,i);
y(:,i) = C*zxx(:,i) + D*u(:,i);
%CONTROLLER STARTS HERE-------------------------------------
%Observe state
xh(:,i+1) = [x(:,i+1); y(:,i)];
zxh(:,i+1) = [Z'*x(:,i+1); y(:,i)];
%Solve dynamic optimisation problem
f = fd*[Z'*xh(1:n_states,i+1);xh(n_states+1:end,i+1)-ref(:,i)];
f(1:n_in) = f(1:n_in)-r*u_old;
U = quadprog(H,f,[],[],[],[],Lb,Ub,[],opt);
u(:,i+1) = U(1:n_in); u_old = u(:,i);
end;

tm=cputime()-tm;
fprintf(1,\n  \t Closed loop simulation took %f sec\n',tm);
toc

% Plots
figure;
plot(t(1:end-1),ref(1:end-1),'r--','linewidth',1.5); hold on
plot(t(1:end-1),y(1:end-1),'k-','linewidth',1); hold off;
ylabel('Output and reference');
xlabel('Time (dimensionless)');
figure;
plot([0 sim_length],[ub ub],'r--'); hold on;
plot([0 sim_length],[lb lb],'r--');
stairs(t(1:end-1),u(:,1:end-1),'b-'); hold off;
xlabel('Time (dimensionless)');
ylabel('Input and constraints');

A.3.3 Utilities: functions used by the previous codes

function [ref]=generate_reference(td,states,params)
step=max(size(params));
r=0;
intx=max(size(states))/2;
intt=100*td+1;
Tc=zeros(step1,intt);
for i=1:intt; Tc(1:step1,i)=params; end
t0=0;
tf=td;
x0=states;
logplot=0;
[FC FT]=Tubular_step(intx,r,Tc,step1,t0,tf,intt,x0,logplot);
n=size(FC,1);
ref=FT(n,:);
end

function [jac gradu vv]=jacobian_nonlinear_eigs(states,params,nb)
global states1 params1
h=le5;
n=max(size(states));
np=max(size(params));
states1=states;
params1=params;
jac=zeros(n,nb);
gradu=zeros(n,np);
end
opts.disp=0;
[V D]=eigs(@jacvecmult,n,nb,'lm',opts);
vv=orth([real(V) imag(V)]);

if tnb<nb
    vv(:,tnb+1)=vv(:,i);
else
    vv=vv(:,i:nb);
end

for i=1:nb-tnb
    vv(:,tnb+i)=vv(:,i);
end

if tnb>nb
    vv=vv(:,1:nb);
end

% Calculate jac
for i=1:nb
    jac(:,i)=jacvecmult(vv(:,i));
end

jac=eye(nb)-vv'*jac;

% Calculate gradu
for i=1:np
    paramsf=params;
    paramsb=params;
    paramsf(i)=params(i)+h;
    paramsb(i)=params(i)-h;
    ff=time_integration(0.1,states,paramsf);
    fb=time_integration(0.1,states,paramsb);
    gradu(:,i)=(ff-fb)/(2*h);
end

gradu=vv'*gradu;

function [Lamda Phi]=largematrices(N,M,n_in,n_out,nb,A,B,C,D)
    Phi = zeros((N+1)*n_out);
    AB = zeros((N+1)*n_out,n_in);
    AB(1:n_out,:) = D;
    Lamda(1:n_out,:) = C;
    AA=eye(nb);
    for j = 2:N+1,
        AB((j-1)*n_out+1:(j)*n_out,:) = C*AA*B; %C*A^(j-2)*B;
        AA=AA*A;
        Lamda(1+(j-1)*n_out:j*n_out,:) = C*AA; %C*A^(j-1);
    end;
    for j = 1:M+1,
        Phi(1+(j-1)*n_out:end,1+(j-1)*n_in:(j)*n_in) = AB(1:(N-j+2)*n_out,:);
    end;
    Phi = sparse(Phi);
end

function [A,B,C,D,Z]=linearise_nonlinear_cstr(Td,xnom,params,nb)
    n=max(size(xnom));
    np=max(size(params));
    [A B Z]=jacobian_nonlinear_eigs(xnom,params,nb);
    m=size(A,1);
    C=zeros(1,n);
    C(n)=1;
    C=(Z'*C')';
    D=zeros(1,np);
    % Discretization
    sysc=ss(A,B,C,D);
    sysd=c2d(sysc,Td,'zoh');
    [A,B,C,D]=ssdata(sysd);
end
function newstatesvector = time_integration(td, states, params)
    step1 = max(size(params));
    r = 0.5;
    intx = max(size(states))/2;
    intt = 100*td+1;
    Tc = zeros(step1, intt);
    for i = 1:intt; Tc(1:step1,i)=params; end
    t0 = 0;
    tf = td;
    x0 = states;
    logplot = 0;
    [FC FT] = Tubular_step(intx, r, Tc, step1, t0, tf, intt, x0, logplot);
    n = size(FC,2);
    newstatesvector = [FC(:,n); FT(:,n)];
end

function [T Y] = time_profile(sim_time, init_cond, params)
    step1 = 8;
    r = 0.5;
    intx = 16;
    intt = 1001;
    Tc = zeros(step1, intt);
    for i = 1:intt; Tc(1:step1,i)=params; end
    t0 = 0;
    tf = sim_time;
    logplot = 0;
    [FC FT] = Tubular_step(intx, r, Tc, step1, t0, tf, intt, init_cond, logplot);
    T = 0:sim_time/(intt-1):sim_time;
    T = T';
    Y = [FC; FT];
    Y = Y';
end

A.3.4 Simulator

function [FC FT] = Tubular_step(intx, r, Tc, step1, t0, tf, intt, x0, logplot)
    time = cputime();
    tx0 = 0;
    tx1 = 1;
    nk = intx;
    xx = linspace(tx0, tx1, intx);
    tt0 = t0;
    ttl = tf;
    tt = linspace(tt0, ttl, intt);
    m = length(tt);
    dt = (tt(m) - tt(1))/(m-1);
    % Constants' list
    C0 = 0;
    T0 = 0;
    iwt = 1;%256=2^8
    Pec = 7.0;
    Pet = 7.0;
    Bc = 0.1;
    Bt = 2.5;
    gamma = 10.0;
    betat = 2.0;
dx(1:nk)=1/(nk-1);
n=m;
itpn=n;
itp=itpn;

if (size(Tc,2)==1)  
   for i=1:itpn  
      Tc(:,i)=Tc(:,1);  \% if dim(Tc)=1 then consider it constant  
   end  \% for the time interval considered for the  
end  \% simulation  

imt=1;  \%linear FEM method, if ntr_p=3 then quadratic FEM method  
ntr_p=2;  
for i=1:(nk-1)  
ic(i,1)=i;  
ic(i,2)=i+1;  
end  

bn(1:nk,1:nk)=0;  
b(1,1)=1;

\% Enter Gauss points and weigths for integration from -1 to 1  
\% Fifth order - three Gauss points (values from Table)  
\%
\% gp(1)=(3/5)^0.5;  
gp(2)=0;  
gp(3)=(3/5)^0.5;  
wd(1)=5/9;  
w(2)=8/9;  
w(3)=5/9;  
for k=1:3  
gp(k)=(gp(k)+1)/2;  
w(2)=wd(k)/2;  
end  
for im=1:imt  
   for iw=1:iwt  
      for nt=1:nk  
         if(jt==1)  
            FC(nt,jt)=x0(nt);  \%0;  
            FT(nt,jt)=x0(nt+nk);  \%0;  
         elseif(jt>1)  
            FC(nt,jt)=FC(nt,jt-1);  
            FT(nt,jt)=FT(nt,jt-1);  
         end  
      end  
   end  
\% \% NEWTON'S METHOD  
\%--------------------------------------------------------------  
iter=0;  
err=0;  
tol=1.0e-05;  
while ((err>tol) | (iter==0)) & (iter<500)  
   err=0;  
   iter=iter+1;  
   rv(1:2*nk)=0;  
   f(1:nk)=0;  
   ft(1:nk)=0;  
   cj(1:2*nk,1:2*nk)=0;  
   rvn(1:nk,1:nk)=0;  
   b(1:nk,1:nk)=0;  
   for i=1:(nk-1)  
      for j=1:3  
         \% \% SOLUTION USING FINITE ELEMENTS  
         \% \% Transform Gauss points and weights for integration from 0 to 1
% (this is done for integrating over a local element)
% z=gp(j);
% [phi,phix]=tfunc(z,ntr_p);
for l=1:ntr_p
  il=ic(i,l);
  for m=1:ntr_p
    i2=lc(l,m);
    % assembly of matrix rvn

    rvn(il,i2)=rvn(il,i2)+wd(j)*(phix(l)*phix(m)/dx(il));
  end % m
end % l
end % i
end % j

for i=1:nk
  for j=1:nk
    kt=ceil(j/(nk/step1));
    Tca=Tc(kt,jt);
    if (jt==1)
      else
        if ((j==1))
          ca1=Bc*(1+FC(j,jt))*exp(gamma*FT(j,jt)/... 
              (1+FT(j,jt)));
          ca2=-Pec*((1-r)*C0+r*FC(nk,jt)-FC(1,jt));
          rv(i)=rv(i)+b(i,j)*ca2-bn(i,j)*... 
            (0-1/Pec*ca2)+rvn(i,j)*1/Pec*FC(j,jt)-... 
            FC(j,jt-1))/dt;
        c(j,j)=c(j,j)+b(i,j)*... 
          exp(gamma*FT(j,jt)/(1+FT(j,jt)))*... 
          FT(j,jt)/(1+FT(j,jt))^2*1);
        ta1=Bt*Bc*(1+FC(j,jt))*exp(gamma*... 
          FT(j,jt)/(1+FT(j,jt)));
        ta2=-Pet*((1-r)*T0+r*FT(nk,jt)-FT(1,jt));
        rv(i+nk)=rv(i+nk)+b(i,j)*ta2-bn(i,j)*... 
          (0-1/Pet*ta2)+rvn(i,j)*1/Pet*... 
          FT(j,jt)-b(i,j)*... 
          (Tca-FT(j,jt)))*... 
          b(i,j)+b(i,j)*... 
          (FT(j,jt)^(1+FT(j,jt)))+... 
          b(i,j)^*1/dt;
      end % j
    end % i
    elseif((j>1) && (j<nk))
      ca1=Bc*(1+FC(j,jt))*exp(gamma*FT(j,jt)/... 
        (1+FT(j,jt)));
      ca2=-Pec*((1-r)*C0+r*FC(nk,jt)-FC(1,jt));
      rv(i)=rv(i)+b(i,j)*(FC(j,jt)-... 
        FC(j,jt-1))/dx(il);
appendix: source code

\[
\begin{align*}
    &cj(i,j) = cj(i,j) + b(i,j) \cdot (1/dx(j)) - \ldots \\
    &bn(i,j)*(-1/Pec*(-1)*Pec*(-1)) + \ldots \\
    &rvn(i,j)*/1/Pec*1*b(i,j)*Bc* \ldots \\
    &exp(gamma*FT(j,jt)/(1+FT(j,jt)))* \ldots \\
    &l+b(i,j)*1/dt; \\
    &cj(i,j+nk) = cj(i,j+nk) + b(i,j)*Bc*(1+ \ldots \\
    &FC(j,jt)) * exp(gamma*FT(j,jt)/(1+ \ldots \\
    &FT(j,jt)))*gamma*(1/(1+FT(j,jt)) = \ldots \\
    &FT(j,jt)/(1+FT(j,jt)))*2*1); \\
    &tal = Bt*Bc*\ldots \\
    &FT(j,jt)/(1+FT(j,jt))); \\
    &ta2 = Pet*((1-r)*T0+r*FT(nk,jt)-FT(1,jt)); \\
    &rv(i+nk) = rv(i+nk) + b(i,j)*\ldots \\
    &FT(j,jt)/dx(j) - bn(i,j)*(-1/Pet* \ldots \\
    &ta2 + rvn(i,j)*1/Pet*FT(j,jt) - \ldots \\
    &b(i,j)*tal-b(i,j)*betat*(Tca= \ldots \\
    &PT(j,jt)) + b(i,j)*FT(j,jt) = \ldots \\
    &PT(j,jt-1))/dt; \\
    &cj(i+nk,j) = cj(i+nk,j) - b(i,j)*Bt*Bc* \ldots \\
    &exp(gamma*PT(j,jt)/(1+PT(j,jt))); \\
    &cj(i+nk,j+nk) = cj(i+nk,j+nk) + b(i,j)* \ldots \\
    -(1/dx(j)) - bn(i,j)*(-1/Pet* \ldots \\
    &Pet*(-1)) + \ldots \\
    &rnv(i,j)*1/Pet*1*b(i,j)* \ldots \\
    &Bt*Bc*(1+FC(j,jt)))*exp(gamma* \ldots \\
    &PT(j,jt)/(1+PT(j,jt)))*gamma*(1/ \ldots \\
    &(1+PT(j,jt))-FT(j,jt)/(1+PT(j,jt)))* \ldots \\
    &^2*1)-b(i,j)*betat*(-1)+b(i,j)* \ldots \\
    &b(i,j)*1/dt; \\
    &elseif(j==nk) \\
    &ca1 = Bc*(1+FC(j,jt)) * \ldots \\
    &exp(gamma*PT(j,jt)/(1+PT(j,jt))); \\
    &ca2 = 0; \\
    &rv(i) = rv(i) + b(i,j)*0 - \ldots \\
    &bn(i,j)*0 - b(i,j)* \ldots \\
    &ca2 + rvn(i,j)*1/Pec*FC(j,jt) + \ldots \\
    &b(i,j)*ca1+b(i,j)*FC(j,jt) - \ldots \\
    &FC(j,jt-1))/dt; \\
    &cj(i,j) = cj(i,j) + b(i,j)*0 - bn(i,j)*(-1/ \ldots \\
    &Pec*0) + \ldots \\
    &rnv(i,j)*1/Pec*1*b(i,j)*Bc* \ldots \\
    &exp(gamma*PT(j,jt)/(1+PT(j,jt)))* \ldots \\
    &1+b(i,j)*1/dt; \\
    &cj(i,j+nk) = cj(i,j+nk) + b(i,j)* \ldots \\
    -(1/dx(j)) - bn(i,j)*(-1/ \ldots \\
    &Pec*(-1)) + \ldots \\
    &rnv(i,j)*1/Pec*1*b(i,j)*Bc* \ldots \\
    &exp(gamma*PT(j,jt)/(1+PT(j,jt)))* \ldots \\
    &1+b(i,j)*1/dt; \\
    &cj(i+nk,j) = cj(i+nk,j) - b(i,j)*Bt*Bc* \ldots \\
    &exp(gamma*PT(j,jt)/(1+PT(j,jt))); \\
    &cj(i+nk,j+nk) = cj(i+nk,j+nk) + b(i,j)* \ldots \\
    -(1/dx(j)) - bn(i,j)*(-1/ \ldots \\
    &Pec*(-1)) + \ldots \\
    &rnv(i,j)*1/Pec*1*b(i,j)*Bc* \ldots \\
    &exp(gamma*PT(j,jt)/(1+PT(j,jt)))* \ldots \\
    &1+b(i,j)*1/dt; \\
    &end \% if j \\
    &end \% if jt \\
    &end \% j \\
    &end \% if j
Appendix: source code

function [phi,phix]=tfunc(z,ntr_p)
% evaluation of local basis functions and derivatives
% format long
% linear basis functions (ntr=2)--------------------------
if(ntr_p==2)
    phi(1) = 1. - z;
    phi(2) = z;
    phix(1)=-1.0;
    phix(2)= 1.0;
% quadratic basis functions (ntr=3)------------------------
elseif(ntr_p==3)
    phi(1) = 1. - 3.* z + 2.* (z^2);
    phi(2) = 4.* (z - z^2);
    phi(3) =-z + 2.* (z^2);
    phix(1)=-3. + 4.* z;
    phix(2)= 4. - 8.* z;
    phix(3)=-1. + 4.* z;
end

A.3.5 Script for plotting

t = 0:Ts:sim_length;
intx=size(xx,1)/2;
z=0:1/(intx-1):1;
% plot results
figure;
plot(t,ref','r--','Linewidth',2); hold on
plot(t,y','k-','Linewidth',1); hold off;
legend('reference','output')
ylabel('Output and reference');
xlabel('Dimensionless time');
for i=1:n_in
    tt(i,:)=t;
end
figure;
plot(tt',u','Linewidth',2); hold on;
plot(t,ub*ones(1,T),'r--','Linewidth',4);
plot(t,lb*ones(1,T),'r--','Linewidth',4);
xlabel('Dimensionless time');
ylabel('Input and constraints');
for i=1:n_in
    legendtitle(i,:)=char('u     ');
    legendtitle(i,2:2+ceil(log10(1)+1e-16)-1)=char(int2str(i));
end
legendtitle(n_in+1,:)=char('bounds');
legend(legendtitle)
hold off;

% Plot profiles
figure;
surf(t,z,xx(1:intx,:))
shading interp
view(72,27)
xlabel('Time')
ylabel('Length')
zlabel('Concentration')
figure;
surf(t,z,xx(intx+1:end,:))
shading interp
view(72,27)
xlabel('Time')
ylabel('Length')
zlabel('Temperature')