

SLOVAK TECHNICAL UNIVERSITY IN BRATISLAVA
Faculty of Chemical and Food Technology
Institute of Information Engineering, Automation and Mathematics
Radlinského 9, 812 37 Bratislava



Bc. Marián Podmajerský

**PARAMETER ESTIMATION IN PROCESSES
FROM EXPERIMENTAL DATA**

Master Thesis

Supervisors

doc. Dr. Ing. Miroslav Fikar, STU Bratislava
prof. M.A. Latifi, ENSIC Nancy

Consultant

Ing. Michal Čižniar, STU Bratislava

2007

Ústav: **Informatizácie, automatizácie a matematiky**

Oddelenie: **Informatizácie a riadenia procesov**

Číslo: 11/UIAM/2007

Vec: **Zadanie diplomovej práce**

Meno a priezvisko študenta: **Bc. Marián Podmajerský**

Meno a priezvisko vedúceho diplomovej práce: **doc. Dr. Ing. Miroslav Fikar**

Meno a priezvisko konzultanta diplomovej práce: **Ing. Michal Čižniar**

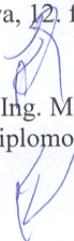
Názov diplomovej práce:

Identifikácia parametrov procesov z experimentálnych údajov

Termín odovzdania diplomovej práce: **19. mája 2007**

Diplomová práca sa odovzdáva v 3 exemplároch vedúcemu ústavu – oddelenia.

Bratislava, 12. februára 2007


doc. Dr. Ing. Miroslav Fikar
vedúci diplomovej práce

doc. Dr. Ing. Miroslav Fikar
riaditeľ ústavu




doc. Dr. Ing. Miroslav Fikar
vedúci oddelenia


prof. Ing. Dušan Bakoš, DrSc.
dekan

First of all, I want to express my sincere gratitude to my thesis supervisor and head of the Institute of Information Engineering, Automation and Mathematics at Faculty of Chemical and Food Technology of the Slovak Technical University in Bratislava, doc. Dr. Ing. Miroslav Fikar, for offered opportunity be a Socrates student at foreign university and his patient guidance throughout my studies.

My big gratification and appreciation goes to my thesis supervisor in Nancy, professor M. Abderazzak Latifi from ENSIC-INPL, that instructs me on research journey and giving me support I need.

Also I would like to express my thank to Ing. Michal Čížniar, for his guidance, comments, tireless support, and debates that inspired me.

Lastly, I would like to thank my family for standing behind me and support my studies in many ways. I am deeply indebted to them and grateful for what they have given me.

Bratislava, Nancy, 2007

Marián Podmajerský

Abstrakt

Táto práca sa zaoberá dynamickou a globálnou optimalizáciou problémov určovania parametrov systémov opísaných sústavou diferenciálno-algebraických rovníc. Určovanie parametrov semi-empirických modelov je dôležité vo veľkej oblasti inžinierstva a aplikovaných vied. Metóda chýb na premenných bola značne študovaná z pohľadu riešenia. Zahŕňa minimalizáciu vážených súm štvorcov odchýliek vzhľadom na rovnice modelu. Bola vyvinutá metóda ortogonálnej kolokácie na konečných prvkoch, pomocou ktorej sa diferenciálne rovnice opisujúce správanie sa dynamického systému konvertujú na systém algebraických rovníc. Táto bola implementovaná do MATLABu ako programový balík DYNOPT. Táto je porovnávaná s integračnou metódou implementovanou vo FORTRANe ako programový balík GDOC.

Abstract

This work deals with dynamic and global optimization for parameter estimation of differential algebraic systems. The estimation of parameters in semi-empirical models is essential in numerous areas of engineering and applied science. The error-in-variables method has been studied from a computational standpoint. This method involves the minimization of a weighted sum of squared errors subject to the model equations. In this work, a method converting the dynamic system of equations into a set of algebraic constraints through the use of orthogonal collocation on finite elements is implemented within MATLAB code DYNOPT. It is compared to a integration method implemented within FORTRAN code GDOC.

Contents

1	Introduction	6
2	Problem Definition	9
2.1	Cost Functional	9
2.1.1	Maximum-Likelihood Estimation	9
2.2	Process Model Equations	12
2.3	Constraints	12
3	Numerical Methods	14
3.1	Sequential Approach	14
3.2	Simultaneous Approach	15
4	Global Optimization	18
4.1	Multistart Method	18
4.2	α BB Method	19
4.2.1	The α BB Algorithm	19
4.2.2	Convex Relaxations	20
4.2.3	Equality Constraints	24
4.2.4	Rigorous Calculation of α	25
4.2.5	Branching Strategies	27
4.2.6	Variable Bound Updates	29
5	Computational Studies	31
5.1	Example 1	31

5.1.1	Problem Formulation	31
5.1.2	Results	32
5.2	Example 2	34
5.2.1	Problem formulation	34
5.2.2	Results	34
5.3	Example 3	37
5.3.1	Problem formulation	37
5.3.2	Results	37
5.4	Example 4	40
5.4.1	Problem formulation	40
5.4.2	Results	40
6	Conclusions	42
	Bibliography	44
	Appendices	48
A	Dynopt GUI and Symdynopt	48
A.1	Example definition	49
A.2	Symdynopt	49
A.3	Dynopt GUI	55
A.3.1	Step 1: Create or Load a Problem	55
A.3.2	Step 2: Optimization Options	55
A.3.3	Step 3: Initialisation Values for Optimization	56
A.3.4	Step 4: Cost and Process	56
A.3.5	Steps 5-7: Constraints	57
A.3.6	Step 8: Save and Solve Problem	57

List of Figures

3.1	Collocation method on finite elements.	15
4.1	Branch-and-Bound Procedure	20
5.1	Example 1: DYNOPT	33
5.2	Example 1: GDOC	33
5.3	Example 2: DYNOPT	36
5.4	Example 2: GDOC	36
5.5	Example 3: DYNOPT	38
5.6	Example 3: GDOC	38
5.7	Example 4: DYNOPT	41
A.1	Optimization type selection	56
A.2	Bundle of initial optimization settings	57
A.3	Form for definition of initials and bounds depends on preliminary selections	58
A.4	Error message report for incorrect inputs	58
A.5	Form for cost function and process model	59
A.6	Equation input dialog	59
A.7	Form for equality and inequality constraints at initial time	60
A.8	Form for equality and inequality constraints between initial and final time	61
A.9	Form for equality and inequality constraints at final time	62
A.10	Form to save and solve problem	63
A.11	Save dynamic optimization problem dialog	63

List of Tables

5.1	Example 1 – Experimental Data	32
5.2	Example 1 – α BB method performed by GDOC	32
5.3	Example 1 – Multistart method with 100 runs performed by DYNOPT (some picked values of objective function)	33
5.4	Example 1 – Multistart method with 100 runs performed by GDOC (some picked values of objective function)	33
5.5	Example 2 – Experimental Data	35
5.6	Example 2 – Multistart method with 100 runs performed by DYNOPT (some picked values of objective function)	35
5.7	Example 2 – Multistart method with 100 runs performed by GDOC (some picked values of objective function)	36
5.8	Example 2 – α BB method performed by GDOC	36
5.9	Example 3 – Experimental Data	38
5.10	Example 3 – Multistart method with 100 runs performed by DYNOPT	38
5.11	Example 3 – Multistart method with 100 runs performed by GDOC (some picked values of objective function)	39
5.12	Example 3 – α BB method performed by GDOC	39
5.13	Example 4 – Multistart method with 100 runs performed by DYNOPT (some picked values of objective function)	41

Abbreviations

α **BB** — α Branch and Bounds

CVP — Control Vector Parametrization

DAE — Differential-Algebraic Equations

DYNOPT — DYNamic OPTimization toolbox for Matlab

GDOC — Routine for Global Optimization of problems with ODE

LB — Lower Bound

NLP — Non-Linear Problem

ODE — Ordinary Differential Equations

SQP — Sequential Quadratic Programming

TP — Total Parametrization

UB — Upper Bound

CHAPTER 1

Introduction

In last years, a noticeable improvement in most of scientific disciplines can be observed, especially rising computing power patches a missing part between possible implementation of algorithms and theoretical basement. That allows to solve more advanced technical problems where parameter estimation belongs with certain. Nowadays, chemical industry include variety of processes, nonlinear in nature. Their dynamic behaviour may be described either by a set of ordinary differential equations (ODE) or by a set of differential-algebraic equations (DAE) comming from the kinetic expressions, with or without qualitative constraints to be taken into account. Frequently, these models contain a big number of unknown parameters *e.g.*, kinetic rate constants, activation energies, growthrates, *etc.*

Firstly we have the kinetic model with several unknown adjustable parameters, which vary with conditions *e.g.*, temperature, pressure, catalyzation, *etc.* Secondly, we have experimental measurements obtained by a laboratory. Then, the objective is to find as good as possible values of model unknowns, by comparing them. The comparison is defined as the minimization of error between data produced by experiment and data provided by mathematical description. The early methods for parameter estimation were formulated by Bard and Lapidus (1968) with gradient based approach and successfully solved the kinetic constants in different examples. Southwell (1969) demonstrated a calculation method derived from gradient based approach for models with linear combination of two parameters. He calculated through the use of iterative matrix calculation. Schwetlick and Tiller (1985) exploited the structure of the Jacobian matrix and developed new correction methods, that extended approach of Bard and Lapidus (1968) for models with more

than two variables and highly nonlinear in parameters. Britt and Luecke (1973) defined additional equality constraints and used Lagrange multipliers and achieved constraint linearization. Anderson et al. (1975) also present successive linearization approach in the determination of model parameters. Bellman et al. (1967) presented sequential algorithm for solving differential system using quasi-linearization. Algorithm which uses sensitivities to predict states with respect to the parameters proposed by Hwang and Seinfeld (1972). They also introduced problem of many local minima and defined new objective function weight, that should rise the chance in the global optimum searching manner. Kalogerakis and Luus (1983) introduced algorithm for direct search method to improve initial guess for a quasi-linearization or Gauss-Newton type minimization. Luus (1998) modified direct search for problem with larger count of local minima in Lotka-Volterra problem. Many researchers refer to approach that converse the dynamic system into a set of algebraic equations. Wide range of polynomial approximations to solve differential problems was offered by Villadsen and Michelsen (1978). Van Den Broesch and Hellinckx (1974) used collocation techniques combined with linearization. Tjoa and Biegler (1991) present algorithm based on orthogonal collocation on finite elements. Liebman et al. (1992) used same discretization method and added a general SQP approach to estimate parameters. Esposito and Floudas (2000) presented the use of αBB global optimization searching method with total parametrization and control vector parametrization algorithms to solve parameter estimation problems.

In particular, one method, *maximum likelihood* (first published in Bard (1974)), has been presented extensively in the the literature. It seems to be an effective way to locate solution for unknown parameters in dynamic nonlinear models. There are several formulation of this method. One, popular, called also *error-in-variables* approach, assumes that error is normally distributed over time horizon with a zero mean and existing covariance matrix is diagonal. Said by another words, the squared difference between the observations and the predictions is minimized along time. The difficulty, which is often discussed and has been more important subject of research, is type of numerical method and its quality. We know several techniques how to find solution for the unconstrained and constrained nonlinear differential algebraic equations. The first technique uses integration routines as well as sensitivities to determine states of dynamic model. This technique is sequential, since minimization of the criteria and solving of differential algebraic system are done by sequential manner. The method was presented by Chen and Hwang (1990); Goh and Teo (1988a) and it is called as Control Vector Parametrization (CVP) method. The second technique is simultaneous and was introduced by Cuthrell

and Biegler (1987, 1989); Logsdon and Biegler (1989). It is called Total Parametrization (TP) method, because both control and state vector is discretized and approximated by a set of polynomials. Orthogonal collocation is simultaneous method and approximates differential system by Lagrange polynomial with Legendre roots. This technique converts dynamic system into a set of algebraic equations resulting in a non-linear programming (NLP) problem. For NLP problems, there are standard and specially designed solvers.

Dynamic optimization solvers increase the precision of model prediction, what is in case of squared error noticeable improvement. The cost function based on difference, at appropriate times, between model and measurements, can produce more than one locally optimal solution. How many they are, depends on influence the error to true value. Several approaches were developed for determination of the best possible fits. Each is very costly in matter of computational effort. There are two logical ways how to achieve a better solution: the first one is to run optimization with different initial guesses through investigated interval for each parameter that is called multistart method. Every solution for that problems can achieve only the closest optimum to initial guess. So, enough initial guesses can locate all minima in range. On the other side, this approach provides no guarantee that the lowest value is the global solution for parameters. The second way is to search through range within algorithm not based on statistics but with strict rules. The branch-and-bound algorithm searches within redundant branching of intervals and at the end locates ϵ -global solution. The guarantee secures an underestimating of terms. The linear terms do not need to underestimate, and any other need to. There are underestimators for bilinear, trilinear, fractional terms, and also general non-convex underestimators based on coefficient α . This method is named α BB global optimization method and insures the guarantee of the global solution within ϵ precision. When difference between underestimated and original term is within ϵ range, this interval is ϵ -global solution. Both methods will be closely described in Chapter 4.

Various examples of parameter estimation will be presented in Chapter 5. Above mentioned methods were verified using multistart implemented in both GDOC (uses CVP method) and DYNOPT (uses TP), and using GDOC with α BB approach. α BB approach was also implemented as a part of DYNOPT bundle, but only for statical problems. Results reported in Chapter 5 are comparison of two techniques: TP (implemented within DYNOPT) and CVP (implemented within GDOC), but only in case of multistart. In case of GDOC, it is not exact α BB approach, however the determination of proper α values depends on Hessian of the optimization problem. When the sequential approach is used, Hessian matrix is unavailable in explicit form.

Problem Definition

The aim of this chapter is to define dynamic optimization problem (DOP) for parameter estimation for differential-algebraic systems.

2.1 Cost Functional

Singer et al. (2006) used error-in-variables formulation to estimate the parameters of the given ODE/DAE model described by equation (2.14). The objective is to minimize the weighted squared error between the observed values $\bar{\mathbf{x}}$ and those predicted by the model $\hat{\mathbf{x}}$. All of the measured variables are included in the objective function as follows:

$$\min_{\hat{\mathbf{x}}, \mathbf{p}} \sum_{i=1}^{n_m} (\hat{\mathbf{x}}(t_i) - \bar{\mathbf{x}}(t_i))^T (\hat{\mathbf{x}}(t_i) - \bar{\mathbf{x}}(t_i)) \quad (2.1)$$

where t_i is the time associated with the i th data point from the set of n_m measurements taken over the time horizon $t \in [t_0, t_f]$.

2.1.1 Maximum-Likelihood Estimation

As it was denoted, the objective is to minimize weighted squared error between the observations and predictions. The predictions are performance indicis or solutions of states of mathematic model, at appropriate measurement times. We can define them as

$$\mathbf{f}(\mathbf{p}, \mathbf{x}) = \mathbf{0} \quad (2.2)$$

where $\mathbf{x} \in \mathcal{R}_{n_{x^m}}$ is a vector of n_{x^m} experimentally measured and in this case also optimized variables, $\mathbf{p} \in \mathcal{R}_{n_p}$ is a vector of n_p unknown parameters, and $\mathbf{f} \in \mathcal{R}_{n_{x^m}}$ represents the system of n_{x^m} algebraic functions.

Laboratory measurements are affected to some extent by an error. These are related to the true values as follows:

$$\bar{\mathbf{x}}_i = \hat{\mathbf{x}}_i + \boldsymbol{\epsilon}_i, \quad i = 1, \dots, n_m \quad (2.3)$$

where $\hat{\mathbf{x}}_i$ is vector of unknown true values of the experimentally measured variables, $\bar{\mathbf{x}}_i$, at the i th data point from n_m taken measurements, and $\boldsymbol{\epsilon}_i$ is a vector of appropriate additive error.

The objective is to find such values of parameters that participate and describe the measured data with the biggest probability. The following probability satisfies our requests:

$$L(\mathbf{p}, \boldsymbol{\psi}) \equiv \theta(\mathbf{E}(\mathbf{p})|\boldsymbol{\psi}) \quad (2.4)$$

Here, L represents likelihood function and $\mathbf{E} \equiv [\mathbf{e}_1^T, \mathbf{e}_2^T, \dots, \mathbf{e}_{n_m}^T]$ is vector of errors between observations and predictions from different data points. If we suppose that experimental data are independent, without correlation and identically distributed, then we may write the likelihood function in form

$$L(\mathbf{p}, \boldsymbol{\psi}) = \prod_{i=1}^{n_m} \theta(\mathbf{e}_i(\mathbf{p})|\boldsymbol{\psi}_i) \quad (2.5)$$

Taking logarithms ¹, results in

$$\ln L(\mathbf{p}, \boldsymbol{\psi}) = \sum_{i=1}^{n_m} \ln \theta(\mathbf{e}_i(\mathbf{p})|\boldsymbol{\psi}_i) \quad (2.6)$$

where \mathbf{p} is a set of unknown parameters and $\boldsymbol{\psi}$ is a set of statistical parameters. We defined maximization problem, where we are trying to maximize the occurrence of parameter value through a set of measurements (maximizing of values for \mathbf{p} and $\boldsymbol{\psi}$ which maximize L). Likelihood function L represents probability of the occurrence of given set of statistical parameters $\boldsymbol{\psi}$, in a set of observed errors, \mathbf{E} , through whole data set.

Uncorrelated measurements produce error with normal distribution without mean and covariance matrix \mathbf{V} :

$$\theta(\mathbf{e}|\mathbf{V}) = N(\mathbf{e}|\mathbf{V}) = \frac{2\pi^{-n_{x^m}/2}}{\sqrt{|\mathbf{V}|}} \exp \left[-\frac{1}{2} \mathbf{e}^T \mathbf{V}^{-1} \mathbf{e} \right] \quad (2.7)$$

¹For convenience it is much easier to work with $(\ln L)$ and maximization of $\ln L$ is equivalent to maximization of the original function L

Substitution (2.7) into (2.6) yields the following

$$\ln L = -\frac{n_{x^m}n_m}{2}\ln 2\pi - \frac{1}{2}\sum_{i=1}^{n_m}\ln |\mathbf{V}_i| - \frac{1}{2}\sum_{i=1}^{n_m}\mathbf{e}_i^T\mathbf{V}_i^{-1}\mathbf{e}_i \quad (2.8)$$

Next, when we define $\mathbf{R} = \mathbf{e}^T\mathbf{e}$, (2.8) can be reformulated taking following:

Firstly: covariance matrix is completely known for each experiment:

$$\min \sum_{i=1}^{n_m}\mathbf{R}_i\mathbf{V}_i^{-1} \quad (2.9)$$

Secondly: covariance matrix is same for each experimental point $\mathbf{V}_1 = \mathbf{V}_2 = \dots = \mathbf{V}$, and then

$$\min \sum_{i=1}^{n_m}\mathbf{R}_i\mathbf{V}^{-1} \quad (2.10)$$

Thirdly: covariance matrix \mathbf{V} is diagonal with elements v_i :

$$\min \sum_{i=1}^{n_m}\sum_{j=1}^{n_{x^m}}e_{i,j}^2v_j^{-1} \quad (2.11)$$

where $e_{i,j}^2$ is j th component from the vector \mathbf{e}_i , that is the error associated with the j th variable in the i th experiment.

We will use third form, where the squared standard deviation σ_j^2 of the j th variable can substitute original term v_j . Standard deviation equals to

$$\sigma_j = \left(\sqrt{\frac{1}{N_{\text{measurements}} - 1} \sum_{k=1}^{N_{\text{measurements}}} (\bar{x}_{jk} - \bar{x}_j)^2} \right) \quad (2.12)$$

where \bar{x}_j is average over k replicate experiments, such that $\bar{x}_j = (1/N_{\text{measurements}})\sum \bar{x}_{jk}$. Substituting the definition of $e_{i,j}$ from (2.3) minimization takes the following formulation, known as the error-on-variables:

$$\min_{\hat{\mathbf{x}}_i, \mathbf{p}} \sum_{i=1}^{n_m}\sum_{j=1}^{n_{x^m}}\frac{(\hat{x}_{i,j} - \bar{x}_{i,j})^2}{\sigma_j^2} \quad (2.13a)$$

subject to

$$\mathbf{h}(\hat{\mathbf{x}}_i, \mathbf{p}) = \mathbf{0}, \quad i = 1, \dots, n_m \quad (2.13b)$$

2.2 Process Model Equations

The kinetic model under study may be described either by a set of ordinary differential equations (ODE) or differential-algebraic equations (DAE) of the form

$$\mathbf{M}\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{p}), \quad \mathbf{x}(0) = \mathbf{x}_0(\mathbf{p}) \quad (2.14)$$

where t denotes time from interval $[t_0, t_f]$, \mathbf{M} is a constant mass matrix, \mathbf{f} is vector valued function of right sides of ODE/DAE's with $\mathbf{x} \in \mathcal{R}_{n_x}$ as state variables with constant initial conditions \mathbf{x}_0 , and $\mathbf{p} \in \mathcal{R}_{n_p}$ as a set of time independent parameters to be estimated.

2.3 Constraints

Constraints to be accounted for typically include equality and inequality infinite dimensional, interior-point, and terminal-point constraints (Goh and Teo, 1988b).

(i) Infinite dimensional constraints: $t \in [\tau_1, \tau_2], \quad t_0 \leq \tau_1 < \tau_2 \leq t_f$

$$\mathbf{h}(\mathbf{x}, \mathbf{p}, t) = \mathbf{0} \quad (2.15a)$$

$$\mathbf{g}(\mathbf{x}, \mathbf{p}, t) \geq \mathbf{0} \quad (2.15b)$$

(ii) Interior-point constraints: $\tau \in [t_0, t_f]$

$$\mathbf{h}(\mathbf{x}, \mathbf{p}, \tau) = \mathbf{0} \quad (2.15c)$$

$$\mathbf{g}(\mathbf{x}, \mathbf{p}, \tau) \geq \mathbf{0} \quad (2.15d)$$

(iii) Terminal-point constraints: $t = t_f$

$$\mathbf{h}(\mathbf{x}, \mathbf{p}, t_f) = \mathbf{0} \quad (2.15e)$$

$$\mathbf{g}(\mathbf{x}, \mathbf{p}, t_f) \geq \mathbf{0} \quad (2.15f)$$

Using the definition of the model presented in (2.14), taking the additional constraints (2.15) to be satisfied during the optimization into account and assuming that the variance of the error associated with each measured variable is equal, the resulting optimization problem can be rewritten as

$$\min_{\hat{\mathbf{x}}, \mathbf{p}} \sum_{i=1}^{n_m} (\hat{\mathbf{x}}(t_i) - \bar{\mathbf{x}}(t_i))^T (\hat{\mathbf{x}}(t_i) - \bar{\mathbf{x}}(t_i)) \quad (2.16a)$$

such that

$$M\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{p}), \quad \mathbf{x}(0) = \mathbf{x}_0(\mathbf{p}) \quad (2.16b)$$

$$\mathbf{h}(t, \mathbf{x}(t), \mathbf{p}) = \mathbf{0} \quad (2.16c)$$

$$\mathbf{g}(t, \mathbf{x}(t), \mathbf{p}) \leq \mathbf{0} \quad (2.16d)$$

$$\mathbf{x}(t)^L \leq \mathbf{x}(t) \leq \mathbf{x}(t)^U \quad (2.16e)$$

$$\mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U \quad (2.16f)$$

where superscripts $^L, ^U$ are the lower and upper profile bounds.

The numerical methods used to find a deterministic solution of dynamic optimization problems can be grouped into two categories: indirect methods, based on optimal control theory (Bellman, 1957; Bryson and Ho, 1975; Pontryagin et al., 1964), and direct methods, based on modifying the original optimization problem by discretization of process variables. In this work only direct methods are considered. In this category, there are two strategies: sequential method and simultaneous method. The sequential strategy, often called Control Vector Parameterization (CVP), consists in an approximation of the control trajectory by a function of only a few parameters and leaving the state equations in the form of the original system of Differential-Algebraic Equations (DAE) (Goh and Teo, 1988b). In the simultaneous strategy, both the control and state variables are discretized using polynomials (e.g., Lagrange polynomials) of which the coefficients become the decision variables in a much larger Nonlinear Programming problem (NLP) (Cuthrell and Biegler, 1987).

This chapter describes the NLP formulations for parameter estimation problems of the form (2.16) for aforementioned two direct approaches.

3.1 Sequential Approach

As in the optimization problem (2.16) just state variables, \mathbf{x} , as function of time independent parameters, \mathbf{p} , appear as optimization variables, the resulting optimization problem

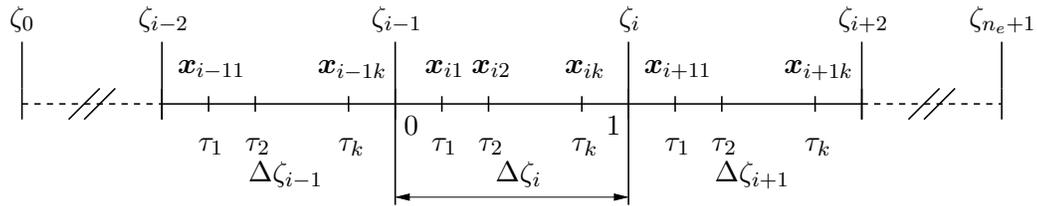


Figure 3.1: Collocation method on finite elements.

can be rewritten as

$$\min_{\mathbf{p}} \sum_{i=1}^{n_m} (\hat{\mathbf{x}}(t_i) - \bar{\mathbf{x}}(t_i))^T (\hat{\mathbf{x}}(t_i) - \bar{\mathbf{x}}(t_i)) \quad (3.1a)$$

such that

$$\mathbf{M}\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{p}), \quad \mathbf{x}(0) = \mathbf{x}_0(\mathbf{p}) \quad (3.1b)$$

$$h_i = G_i(t_j, \mathbf{x}(t_1), \dots, \mathbf{x}(t_f), \mathbf{p}) + \int_{t_0}^{t_f} F_i(t, \mathbf{x}(t), \mathbf{p}) dt = \mathbf{0} \quad (3.1c)$$

$$i = 1, \dots, n_h$$

$$g_k = G_k(t_j, \mathbf{x}(t_1), \dots, \mathbf{x}(t_f), \mathbf{p}) + \int_{t_0}^{t_f} F_k(t, \mathbf{x}(t), \mathbf{p}) dt \leq \mathbf{0} \quad (3.1d)$$

$$k = 1, \dots, n_g$$

$$\mathbf{x}(t)^L \leq \mathbf{x}(t) \leq \mathbf{x}(t)^U \quad (3.1e)$$

$$\mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U \quad (3.1f)$$

where the equality constraints, h , are for $i = 1, \dots, n_h$ and the inequality constraints, g , are for $k = 1, \dots, n_g$. Both, equality and inequality constraints are considered in the Lagrange form (see (Víteček and Vítečková, 2002)).

This approach has been implemented within FORTRAN code GDOC by Singer et al. (2005) and will be used to solve the problems defined in Chapter 5.

3.2 Simultaneous Approach

Many different collocation-based discretisations exist for the solution of ODE/DAE systems. In this section the method of orthogonal collocation on finite elements will be used to parametrize the state variables (Cuthrell and Biegler, 1989; Logsdon and Biegler, 1989, 1992). This transforms the original problem into a fully algebraic NLP.

The state profiles are approximated by piecewise Lagrange polynomials using k point

orthogonal collocation on n_e finite elements as shown in Figure 3.1

$$\mathbf{x}_{k+1}(t) = \sum_{j=0}^k \mathbf{x}_{ij} \phi_j(t) \quad (3.2a)$$

$$\phi_j(t) = \prod_{l=0, l \neq j}^k \frac{(t - t_{il})}{(t_{ij} - t_{il})} \quad (3.2b)$$

for $i = 1, \dots, n_e$, $j = 0, \dots, k$, $l = 0, \dots, k$

These polynomials have the feature that at the time point $t = t_{ij}$, the coefficient of the polynomial, \mathbf{x}_{ij} is the value of the state profile at that point. Therefore, the coefficients have a physical meaning which makes it easier to generate bounds for these variables. Moreover, the times t_{ij} are shifted roots of Legendre polynomial of degree k on interval $[0, 1]$ ($t_{ij} = \zeta_{i-1} + \Delta\zeta_i \tau_j$). Substituing (3.2a) into (2.16b) yields the residual equation

$$\mathbf{M} \sum_{j=0}^k \mathbf{x}_{ij} \dot{\phi}_j(\tau_l) - \Delta\zeta_i \mathbf{f}(t_{il}, \mathbf{x}_{k+1}(t_{il}), \mathbf{p}) = \mathbf{0} \quad (3.3)$$

to be solved at $l = 1, \dots, k$ collocation points on each element $i = 1, \dots, n_e$ with its length $\Delta\zeta_i = \zeta_i - \zeta_{i-1}$, where ζ_{i-1}, ζ_i are the initial and final time of given element, and $t_{il} = \zeta_{i-1} + \Delta\zeta_i \tau_l$ (see Figure 3.1). As the state profile should be continuous over the full time interval $t \in [t_0, t_f]$, continuity constraints of the form

$$\mathbf{x}_{i0} - \sum_{j=0}^k \mathbf{x}_{i-1j} \phi_j(\tau = 1) = \mathbf{0} \quad (3.4)$$

are imposed at each element endpoint (interior knots ζ_i , $i = 1, \dots, n_e$). In addition, the initial conditions are imposed at the start of the first element:

$$\mathbf{x}_{10} - \mathbf{x}(0) = \mathbf{0} \quad (3.5)$$

The algebraic equality (2.16c) and inequality (2.16d) constraints are simply imposed at the collocation points:

$$\mathbf{h}(t_{ij}, \mathbf{x}_{k+1}(t_{ij}), \mathbf{p}) = \mathbf{0} \quad (3.6a)$$

$$\mathbf{g}(t_{ij}, \mathbf{x}_{k+1}(t_{ij}), \mathbf{p}) \leq \mathbf{0} \quad (3.6b)$$

Moreover it is important to determine the values of the state variables $\hat{\mathbf{x}}$ in times t_m , in which the measured data $\bar{\mathbf{x}}$ are taken. This is done by

$$\mathbf{x}_{k+1}(t_m) = \sum_{j=0}^k \mathbf{x}_{ij} \phi_j(\tau_m), \quad \tau_m = \frac{t_m - \zeta_{(i-1)_m}}{\Delta\zeta_{i_m}} \quad (3.7)$$

where the element index i_m , represents the element in which the measurements are taken, defined as $i_m \equiv \{i : \zeta_{(i-1)} \leq t_m \leq \zeta_i\}$.

The problem (2.16) now becomes:

$$\min_{\mathbf{x}_{ij}, \mathbf{p}} \sum_{i=1}^{n_m} (\hat{\mathbf{x}}(t_i) - \bar{\mathbf{x}}(t_i))^2 \quad (3.8a)$$

such that

$$\mathbf{x}_{10} - \mathbf{x}_0 = \mathbf{0} \quad (3.8b)$$

$$\mathbf{M} \sum_{j=0}^k \mathbf{x}_{ij} \dot{\phi}_j(\tau_l) - \Delta \zeta_i \mathbf{f}(t_{il}, \mathbf{x}_{k+1}(t_{il}), \mathbf{p}) = \mathbf{0}, \quad (3.8c)$$

$$i = 1, \dots, n_e, \quad l = 0, \dots, k$$

$$\mathbf{x}_{i0} - \sum_{j=0}^k \mathbf{x}_{i-1j} \phi_j(1) = \mathbf{0}, \quad i = 2, \dots, n_e \quad (3.8d)$$

$$\mathbf{x}_f - \sum_{j=0}^k \mathbf{x}_{n_ej} \phi_j(1) = \mathbf{0} \quad (3.8e)$$

$$\mathbf{h}(t_{ij}, \mathbf{x}_{k+1}(t_{ij}), \mathbf{p}) = \mathbf{0} \quad (3.8f)$$

$$\mathbf{g}(t_{ij}, \mathbf{x}_{k+1}(t_{ij}), \mathbf{p}) \leq \mathbf{0} \quad (3.8g)$$

$$\mathbf{x}_{ij}^L \leq \mathbf{x}_{k+1}(t_{ij}) \leq \mathbf{x}_{ij}^U \quad (3.8h)$$

$$i = 1, \dots, n_e, \quad j = 0, \dots, k$$

$$\mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U \quad (3.8i)$$

where i refers to the element, j , l refer to the collocation points, $\Delta \zeta_i$ represents finite-element lengths, \mathbf{x}_0 holds the values of the states at time $t = 0$, \mathbf{x}_f holds the values of the states at the final time $t = t_f$, \mathbf{h} is vector valued function of the equality constraints evaluated in time t_{ij} , \mathbf{g} is the vector valued function of inequality constraints evaluated in time t_{ij} , and \mathbf{x}_{ij} are the collocation coefficients for the state profiles.

Problem (3.8) can be now solved by any standard nonlinear programming solver.

This approach has been implemented within MATLAB code DYNOPT (Čižniar et al., 2006). This exploits the function *fmincon* as one of several codes included in MATLAB Optimization Toolbox (MathWorks, 2006). DYNOPT will be also used to solve the problems defined in Chapter 5.

Global Optimization

Many of dynamic optimisation problems that are encountered in chemical engineering application are solved today by means of NLP algorithms. However, it is well known that many of them exhibit multiple local optima. This property which can be attributed to nonconvexity of the functions participating in most chemical engineering models, implies that standard local optimization methods will often yield suboptimal solutions and has motivated researchers to develop global optimization algorithms for solving nonconvex NLPs. Furthermore, the steady improvement in the performance of computers constantly extends the scope of problems which are tractable with global optimization approaches.

This chapter describes the multistart method, which is based on statistics, and does not give the guarantee to converge to the global solution and the α BB algorithm (Adjiman et al., 1998a,b), which guarantees the ϵ -convergence to the global solution.

4.1 Multistart Method

Multistart method randomly searches local minima between chosen parameter boundaries. That means, that the optimization is started repeatedly with different set of initials. The results are then compared to each other, and the best solution is chosen. Multistart approach can not provide any guarantee about quality of results.

4.2 α BB Method

Consider one of NLP problems (3.1), or (3.8) which result from applying one of the direct methods (e.g., either CVP or TP) to parameter estimation problem defined by equations (2.16).

$$\min_{\mathbf{z}} \mathcal{J}(\mathbf{z}) \tag{4.1a}$$

s.t.

$$\mathbf{h}(\mathbf{z}) = \mathbf{0} \tag{4.1b}$$

$$\mathbf{g}(\mathbf{z}) \leq \mathbf{0} \tag{4.1c}$$

where $\mathbf{z} \in \mathcal{C} \subseteq \mathcal{R}_{n_z}$ is a vector of optimized parameters of size n_z , and \mathcal{J} represents the optimisation criterion constrained by equality and inequality constraints \mathbf{h} and \mathbf{g} . All of them must belong to $\mathcal{C}^2 \subseteq \mathcal{R}^2$, that means, they have to be a twice-differentiable functions.

4.2.1 The α BB Algorithm

The α BB algorithm operates within a branch-and-bound framework (Adjiman et al., 1998a) and is designed to solve nonconvex minimization problems of the generic type represented by formulation (4.1). The theoretical properties of the algorithm guarantee that such a problem can be solved to global optimality within finite ϵ -convergence.

A branch-and-bound algorithm begins by constructing a relaxation of the original nonconvex problem (4.1). This relaxation is then solved to generate a lower bound on the solution of the original problem and should, in some way be easier to solve than the original problem. In the current context, the relaxation is a convex optimization problem whose objective function underestimates the nonconvex objective function on \mathcal{C} and whose feasible set contains that of the nonconvex problem. This can be achieved by constructing functions that are *convex relaxations* of the objective and constraint functions on \mathcal{C} and formulating a convex optimization problem for these relaxed functions. In addition, because every local minimum of a convex optimization problem is a global minimum, standard NLP algorithms designed to locate local minima can find this lower bound reliably. An upper bound is generated by the value of the nonconvex objective function at any feasible point (e.g., a local minimum found by standard NLP algorithm, or a problem (4.1) evaluation at the solution of the relaxation problem). If these bounds are not within some ϵ tolerance a branching heuristic is used to partition the set \mathcal{C} into two new subproblems (e.g., bisect on one of the variables). Relaxations can be constructed on these

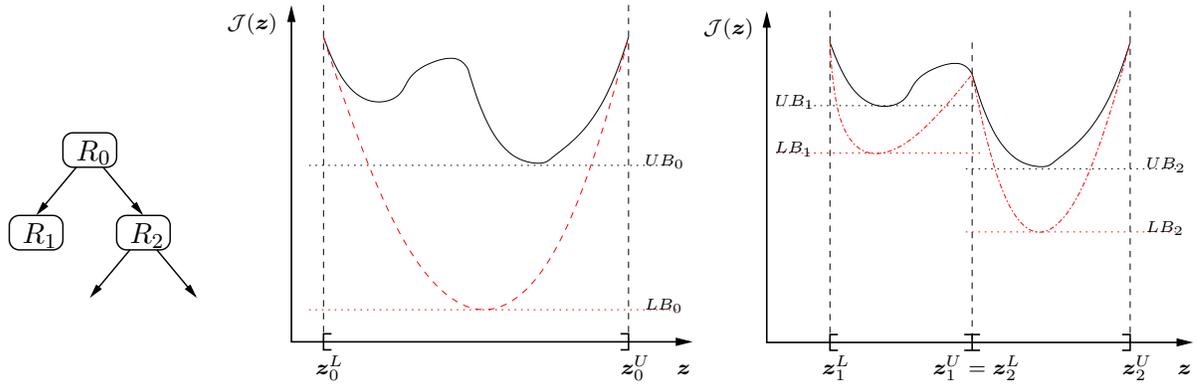


Figure 4.1: The Branch-and-Bound Procedure.

two smaller sets, and lower and upper bound can be computed for these partitions. If the lower bound on a partition is greater than the current best upper bound, a global solution cannot exist in that partition and this partition is excluded from further consideration (*fathoming*). This process of branching, bounding and fathoming continues until the lower bound on all active partitions is within ϵ -tolerance of the current best upper bound. The branch-and-bound procedure is illustrated in Figure 4.1. A convex relaxation (dashed line) of a nonconvex function $J(\mathbf{z})$ is solved on R_0 (center plot) to find a lower bound LB_0 to $J(\mathbf{z})$. Upper bound UB_0 is given by solving the nonconvex function on R_0 . The region R_0 is then subdivided to form regions R_1 and R_2 , and the relaxation is repeated to find a new lower bounds for each region (right plot). By finding a point z in region R_2 where $\mathcal{J}(z)$ is less than the convex lower bound LB_1 for region R_1 , one can show that a global minimum cannot exist in region R_1 . This region can now be discarded from the search tree, whereas R_2 is further subdivided as shown in the branch-and-bound tree (left plot).

4.2.2 Convex Relaxations

A determining step in the convexification strategy is the decomposition of each nonlinear function into a sum of nonconvex terms of special type (e.g., linear, bilinear, trilinear, fractional, fractional trilinear, convex, univariate concave) and nonconvex terms of arbitrary type. Based on these categories, a function $\mathcal{J}(\mathbf{z})$ with continuous second-order derivatives can be written as:

$$\begin{aligned}
 \mathcal{J}(\mathbf{z}) = & LT(\mathbf{z}) + CT(\mathbf{z}) + \sum_{i=1}^{n_{BT}} b_i z_{BT_i,1} z_{BT_i,2} + \sum_{i=1}^{n_{TT}} t_i z_{TT_i,1} z_{TT_i,2} z_{TT_i,3} \\
 & + \sum_{i=1}^{n_{FT}} f_i \frac{z_{FT_i,1}}{z_{FT_i,2}} + \sum_{i=1}^{n_{FTT}} ft_i \frac{z_{FTT_i,1} z_{FTT_i,2}}{z_{FTT_i,3}} + \sum_{i=1}^{n_{UT}} UT_i(\mathbf{z}_{UT_i}) + \sum_{i=1}^{n_{NT}} NT_i(\mathbf{z}) \quad (4.2)
 \end{aligned}$$

where $LT(\mathbf{z})$ is a linear term; $CT(\mathbf{z})$ is a convex term; n_{BT} is the number of bilinear terms, $z_{BT_i,1}$ and $z_{BT_i,2}$ denote the two variables that participate in the i th bilinear term and b_i is its coefficient; n_{TT} is the number of trilinear terms, $z_{TT_i,1}$, $z_{TT_i,2}$ and $z_{TT_i,3}$ denote the three variables that participate in the i th trilinear term and t_i is its coefficient; n_{FT} is the number of fractional terms, $z_{FT_i,1}$ and $z_{FT_i,2}$ denote the two variables that participate in the i th fractional term and f_i is its coefficient; n_{FTT} is the number of fractional trilinear terms, $z_{FTT_i,1}$, $z_{FTT_i,2}$ and $z_{FTT_i,3}$ denote the three variables that participate in the i th fractional trilinear term and ft_i is its coefficient; n_{UT} is the number of univariate concave terms, $UT_i(\mathbf{z}_i)$ is the i th univariate concave term, z_{UT_i} denotes the variable that participates in UT_i ; n_{NT} is the number of general nonconvex terms, $NT_i(\mathbf{z})$ is the i th general nonconvex term. Although it is possible to decompose the nonconvex function into other mathematical structures such as signomial expressions (SE), they are not considered in this work. A detailed description of the treatment of such terms can be found in Maranas and Floudas (1997).

Techniques can be derived to generate valid and, in some cases, very tight convex underestimators of these terms. A detailed description of the treatment of such terms can be found in Adjiman et al. (1998b); Androulakis et al. (1995). In constructing a convex underestimator for the overall function, it is first noted that linear and convex terms do not require any transformation. The special type of nonconvex terms (bilinear, trilinear, fractional, trilinear fractional, and univariate concave terms) is then replaced by very tight convex underestimators which are already known (Adjiman et al., 1998b; Floudas, 2000). The convex envelopes can be constructed by the following simple rules.

Underestimating bilinear terms

In the case of bilinear term xy , a tight convex lower bound over the domain $[x^L, x^U] \times [y^L, y^U]$ is obtained by introducing a new variable w_{BT} which replaces every occurrence of xy in the problem and adding the following four linear inequality constraints:

$$w_{BT} \geq x^L y + xy^L - x^L y^L \quad (4.3a)$$

$$w_{BT} \geq x^U y + xy^U - x^U y^U \quad (4.3b)$$

$$w_{BT} \leq x^L y + xy^U - x^L y^U \quad (4.3c)$$

$$w_{BT} \leq x^U y + xy^L - x^U y^L \quad (4.3d)$$

Underestimating trilinear terms

Any trilinear term of the form xyz can be underestimated over the domain $[x^L, x^U] \times [y^L, y^U] \times [z^L, z^U]$ by introducing a new variable w_{TT} and bounding it by the following linear inequality constraints:

$$w_{TT} \geq xy^L z^L + x^L y z^L + x^L y^L z - 2x^L y^L z^L \quad (4.4a)$$

$$w_{TT} \geq xy^U z^U + x^U y z^L + x^U y^L z - x^U y^L z^L - x^U y^U z^U \quad (4.4b)$$

$$w_{TT} \geq xy^L z^L + x^L y z^U + x^L y^U z - x^L y^U z^U - x^L y^L z^L \quad (4.4c)$$

$$w_{TT} \geq xy^U z^L + x^U y z^U + x^L y^U z - x^L y^U z^L - x^U y^U z^U \quad (4.4d)$$

$$w_{TT} \geq xy^L z^U + x^L y z^L + x^U y^L z - x^U y^L z^U - x^L y^L z^L \quad (4.4e)$$

$$w_{TT} \geq xy^L z^U + x^L y z^U + x^U y^U z - x^L y^L z^U - x^U y^U z^U \quad (4.4f)$$

$$w_{TT} \geq xy^U z^L + x^U y z^L + x^L y^L z - x^U y^U z^L - x^L y^L z^L \quad (4.4g)$$

$$w_{TT} \geq xy^U z^U + x^U y z^U + x^U y^U z - 2x^U y^U z^U \quad (4.4h)$$

Underestimating fractional terms

Fractional terms of the form $\frac{x}{y}$ are underestimated by introducing a new variable w_{FT} and two new inequality constraints which depend on the sign of the bounds on x :

$$w_{FT} \geq \begin{cases} \frac{x^L}{y} + \frac{x}{y^U} - \frac{x^L}{y^U} & \text{if } x^L \geq 0 \\ -\frac{x}{y^U} - \frac{x^L y}{y^L y^U} + \frac{x^L}{y^U} & \text{if } x^L < 0 \end{cases} \quad (4.5a)$$

$$w_{FT} \geq \begin{cases} \frac{x^U}{y} + \frac{x}{y^L} - \frac{x^U}{y^L} & \text{if } x^U \geq 0 \\ -\frac{x}{y^L} - \frac{x^U y}{y^L y^U} + \frac{x^U}{y^U} & \text{if } x^U < 0 \end{cases} \quad (4.5b)$$

Underestimating fractional trilinear terms

For fractional trilinear term of the form $\frac{xy}{z}$, an underestimator is derived by introducing a new variable w_{FTT} by the following inequality constraints for $x^L, y^L, z^L \geq 0$:

$$w_{FTT} \geq \frac{xy^L}{z^U} + \frac{x^L y}{z^U} + \frac{x^L y^L}{z} - 2\frac{x^L y^L}{z^U} \quad (4.6a)$$

$$w_{FTT} \geq \frac{xy^L}{z^U} + \frac{x^L y}{z^L} + \frac{x^L y^U}{z} - \frac{x^L y^U}{z^L} - \frac{x^L y^L}{z^U} \quad (4.6b)$$

$$w_{FTT} \geq \frac{xy^U}{z^L} + \frac{x^U y}{z^U} + \frac{x^U y^L}{z} - \frac{x^U y^L}{z^U} - \frac{x^U y^U}{z^L} \quad (4.6c)$$

$$(4.6d)$$

$$w_{FTT} \geq \frac{xy^U}{z^U} + \frac{x^U y}{z^L} + \frac{x^L y^U}{z} - \frac{x^L y^U}{z^U} - \frac{x^U y^U}{z^L} \quad (4.6e)$$

$$w_{FTT} \geq \frac{xy^L}{z^U} + \frac{x^L y}{z^L} + \frac{x^U y^L}{z} - \frac{x^U y^L}{z^L} - \frac{x^L y^L}{z^U} \quad (4.6f)$$

$$w_{FTT} \geq \frac{xy^U}{z^U} + \frac{x^U y}{z^L} + \frac{x^L y^U}{z} - \frac{x^L y^U}{z^U} - \frac{x^U y^U}{z^L} \quad (4.6g)$$

$$w_{FTT} \geq \frac{xy^L}{z^U} + \frac{x^L y}{z^L} + \frac{x^U y^L}{z} - \frac{x^U y^L}{z^L} - \frac{x^L y^L}{z^U} \quad (4.6h)$$

$$w_{FTT} \geq \frac{xy^U}{z^L} + \frac{x^U y}{z^L} + \frac{x^U y^U}{z} - 2\frac{x^U y^U}{z^L} \quad (4.6i)$$

Underestimating univariate concave terms

Univariate concave terms are trivially underestimated by their linearisation at the lower bound of the variable range. Thus the convex envelope of the concave function $UT(x)$ over $[x^L, x^U]$ is the following linear function of x :

$$UT(x^L) + \frac{UT(x^U) - UT(x^L)}{x^U - x^L}(x - x^L) \quad (4.7)$$

Underestimating nonconvex terms of arbitrary type

For the nonconvex terms of arbitrary type, whose convex envelopes are not known, a convex underestimator is generated by adding to them the relaxation function (Adjiman et al., 1998b), $RF(\mathbf{z}, \boldsymbol{\alpha})$:

$$RF(\mathbf{z}, \boldsymbol{\alpha}) = - \sum_{j=1}^{n_z} \alpha_j (z_j^U - z_j)(z_j - z_j^L) \quad (4.8)$$

where $\alpha_j \geq 0$, corresponds to the variable $j = 1, \dots, n_z$. Given sufficiently large values of the α_j parameters, all nonconvexities in the original function $\mathcal{J}(\mathbf{z})$ can be overpowered by the convex quadratic term and $\mathcal{L}_{\alpha\text{BB}}(\mathbf{z}, \boldsymbol{\alpha})$ is therefore a valid convex underestimator of the form:

$$\mathcal{L}_{\alpha\text{BB}}(\mathbf{z}, \boldsymbol{\alpha}) = \mathcal{J}(\mathbf{z}) + \sum_{i=1}^{n_{NT}} RF_i(\mathbf{z}, \boldsymbol{\alpha}_i) \quad (4.9)$$

where $\boldsymbol{\alpha}_i$ corresponds to the term $i = 1, \dots, n_{NT}$. To derive a valid convex underestimator therefore consists in generating a set of α parameters. How to generate them will be discussed in section 4.2.4.

Overall Convex Underestimator

Based on the decomposition approach given in equation (4.2) and every customized underestimators discussed in sections above, the corresponding lower bounding function $\mathcal{L}(\mathbf{z})$ to $\mathcal{J}(\mathbf{z})$ is:

$$\begin{aligned}
\mathcal{L}(\mathbf{z}) &= LT(\mathbf{z}) + CT(\mathbf{z}) + \sum_{i=1}^{n_{BT}} b_i w_{BT_i} + \sum_{i=1}^{n_{TT}} t_i w_{TT_i} + \sum_{i=1}^{n_{FT}} f_i w_{FT_i} + \sum_{i=1}^{n_{FTT}} ft_i w_{FTT_i} \\
&+ \sum_{i=1}^{n_{UT}} \left[UT_i(z_{UT_i}^L) + \frac{UT_i(z_{UT_i}^U) - UT_i(z_{UT_i}^L)}{z_{UT_i}^U - z_{UT_i}^L} (z_i - z_{UT_i}^L) \right] \\
&+ \sum_{i=1}^{n_{NT}} \left[NT_i(\mathbf{z}) + \sum_{j=1}^{n_z} \alpha_{ij} (z_j^U - z_j)(z_j - z_j^L) \right]
\end{aligned} \tag{4.10}$$

where α_{ij} corresponds to the i -th general nonconvex term and the j -th variable; the w_{BT_i} , w_{TT_i} , w_{FT_i} , w_{FTT_i} must satisfy constraints of the forms given by the sets of equation (4.3), (4.4), (4.5), and (4.6) respectively.

4.2.3 Equality Constraints

In order to generate a valid lower bound on the global solution of the nonconvex NLP, the underestimating NLP generated in each subdomain must be convex. This implies that all inequality constraints in the lower bounding problem should be convex, all equality constraints should be linear and that the size of the feasible region must be increased relative to that of the original nonconvex problem.

Strategies that can be used to underestimate a nonlinear equality constraint depend on the type of terms it involves.

- For equalities in which only linear, bilinear, trilinear, fractional and fractional trilinear terms appear, the nonlinear terms are replaced by new variables which participate linearly in the problem. The equality resulting from the substitution is therefore linear. Moreover, since the set of values these new variables can take on is a superset of the values that can be attained by the nonlinear term, the linear equality corresponds to an enlarged feasible region. Thus given the equality:

$$\begin{aligned}
0 &= LT(\mathbf{z}) + \sum_{i=1}^{n_{BT}} b_i z_{BT_i,1} z_{BT_i,2} + \sum_{i=1}^{n_{TT}} t_i z_{TT_i,1} z_{TT_i,2} z_{TT_i,3} \\
&+ \sum_{i=1}^{n_{FT}} f_i \frac{z_{FT_i,1}}{z_{FT_i,2}} + \sum_{i=1}^{n_{FTT}} ft_i \frac{z_{FTT_i,1} z_{FTT_i,2}}{z_{FTT_i,3}}
\end{aligned} \tag{4.11}$$

the following underestimator can be used:

$$0 = LT(\mathbf{z}) + \sum_{i=1}^{n_{BT}} b_i w_{BT_i} + \sum_{i=1}^{n_{TT}} t_i w_{TT_i} + \sum_{i=1}^{n_{FT}} f_i w_{FT_i} + \sum_{i=1}^{n_{FTT}} ft_i w_{FTT_i} \tag{4.12}$$

where the notation is as previously defined, and the appropriate inequality constraint for the \mathbf{w} variables are added to the problem.

- For nonlinear equality constraint containing convex or general nonconvex terms, the equality obtained by simple substitution of the corresponding underestimator is nonlinear. Furthermore, if it contains univariate concave terms, it is linear but corresponds to a different feasible region. In the presence of either convex, univariate concave or general nonconvex terms, the original equality $h_j(\mathbf{z}) = 0$ must therefore be split as two inequalities of opposite sign:

$$h_j(\mathbf{z}) \leq 0 \quad (4.13a)$$

$$-h_j(\mathbf{z}) \leq 0 \quad (4.13b)$$

These two inequality constraints must then be underestimated independently. The univariate concave terms appearing in the nonconvex equality become convex terms in one of the two inequalities while the convex terms become concave, and the general nonconvex terms become either concave or remain nonconvex.

4.2.4 Rigorous Calculation of α

Since $\mathcal{L}(\mathbf{z})$ in (4.10) is convex means that its Hessian matrix $\mathbf{H}_{\mathcal{L}}(\mathbf{z})$ is positive semi-definite, a useful convexity condition is derived by noting that $\mathbf{H}_{\mathcal{L}}(\mathbf{z})$ is related to the Hessian matrix $\mathbf{H}_{\mathcal{J}}(\mathbf{z})$ of $\mathcal{J}(\mathbf{z})$ by:

$$\mathbf{H}_{\mathcal{L}}(\mathbf{z}) = \mathbf{H}_{\mathcal{J}}(\mathbf{z}) + 2\mathbf{\Delta} \quad (4.14)$$

where $\mathbf{\Delta}$ is a diagonal matrix whose diagonal elements are the α_i 's and is referred to as the *diagonal shift matrix*. In order to derive a valid convex underestimator, the set of α parameters must satisfy the following theorem:

Theorem 1 $\mathcal{L}(\mathbf{z})$, as defined in (4.10), is convex if $\mathbf{H}_{\mathcal{J}}(\mathbf{z}) + 2\mathbf{\Delta} = \mathbf{H}_{\mathcal{J}}(\mathbf{z}) + 2\text{diag}(\alpha_i)$ is positive semi-definite for all $\mathbf{z} \in [\mathbf{z}^L, \mathbf{z}^U]$.

In recent years, a number of deterministic methods have been devised in order to automatically identify an appropriate diagonal shift matrix, based on interval arithmetic. Two classes of approaches to this problem are considered subsequently: (i) Uniform diagonal shift of the Hessian matrix of $\mathcal{J}(\mathbf{z})$; (ii) Nonuniform diagonal shift of the Hessian matrix of $\mathcal{J}(\mathbf{z})$. For further details refer, for instance, to Adjiman et al. (1998b).

Uniform Diagonal Shift Matrix

For this class of methods, the underestimator is re-formulated using a single α value:

$$\mathcal{L}(\mathbf{z}, \alpha) = \mathcal{J}(\mathbf{z}) + \alpha \sum_{j=1}^{n_z} (z_j^U - z_j)(z_j - z_j^L) \quad (4.15)$$

All non-zero elements of Δ are therefore equal to α . It can be shown that $\mathcal{L}(\mathbf{z})$ is a valid convex underestimator of $\mathcal{J}(\mathbf{z})$ if:

$$\alpha \geq \max\left(0, -\frac{1}{2} \min_{i, \mathbf{z}^L \leq \mathbf{z} \leq \mathbf{z}^U} \lambda_i(\mathbf{z})\right) \quad (4.16)$$

where the λ_i 's are the eigenvalues of $\mathbf{H}_{\mathcal{J}}(\mathbf{z})$.

Considering an interval Hessian matrix $[\mathbf{H}_{\mathcal{J}}] \subseteq \{\mathbf{H}_{\mathcal{J}}(\mathbf{z}), \mathbf{z} \in [\mathbf{z}^L, \mathbf{z}^U]\}$, it can be shown that a sufficient condition for the convexity of $\mathcal{L}(\mathbf{z})$ is given by:

$$\alpha \geq \max\left(0, -\frac{1}{2} \lambda_{\min}([\mathbf{H}_{\mathcal{J}}])\right) \quad (4.17)$$

where $\lambda_{\min}([\mathbf{H}_{\mathcal{J}}])$ is the minimum eigenvalue of the interval matrix family $[\mathbf{H}_{\mathcal{J}}]$. Examples of such class of methods are (Adjiman et al., 1998b):

- Gerschgorin's theorem for interval matrices.
- E-matrix method.
- Mori and Kokame's method.
- The lower bounding Hessian method.
- A method based on the Kharitonov theorem.
- Hertz's method.

Nonuniform Diagonal Shift Matrix

This class of methods allows the calculation of a different α value for each variable in order to construct an underestimator of the form shown in equation (4.10). In this case the non-zero elements of the diagonal shift matrix can no longer be related to the minimum eigenvalue of the interval Hessian matrix $[\mathbf{H}_{\mathcal{J}}]$. Examples of such class of methods are (Adjiman et al., 1998b):

- Scaled Gerschgorin's theorem.
- H-matrix method.
- Minimisation of maximum separation distance.

4.2.5 Branching Strategies

Although it does not present any theoretical difficulties, the branching step of any branch-and-bound algorithm often has significant effect on the rate of convergence. This is especially true for the α BB algorithm since the quality of the underestimator depends on the variable bounds in a variety of ways. For instance, if a variable participates only linearly in the problem, branching on it will not have any effect on the accuracy of the convex lower bounding functions. On the other hand, reducing the range of a variable raised to high power is likely to result in much tighter underestimating problems. To take advantage of these observations, the implementation of the α BB algorithm offers some choice of the branching strategies. Four alternatives are currently available:

Strategy 1

The interval to be bisected is determined through the *least reduced axis* rule (Adjiman et al., 1998b) whose application involves the calculation of a *current range to original range* ratio, denoted r_i , for each variable:

$$r_i = \frac{z_i^U - z_i^L}{z_{i,0}^U - z_{i,0}^L} \quad (4.18)$$

where $z_{i,0}^U$ and $z_{i,0}^L$ are, respectively, the upper and lower bounds on variable z_i at the first node of the branch-and-bound tree, and z_i^U and z_i^L are the upper and lower bounds on the variable z_i at the current node of the tree. The variable with the largest r_i is selected for branching.

Strategy 2

While the second branching option requires additional computational effort, it results in a significant improvement of the convergence time for difficult problems. A different measure μ for each type of the underestimator is considered to facilitate the assessment of the quality of the lower bounding function (Adjiman et al., 1998a; Androulakis et al., 1995). The maximum separation distance between the underestimator and the actual term at the optimal solution of the lower bound problem is one possible indicator of the degree of accuracy achieved in the construction of the convex approximation.

- For the bilinear term xy , the maximum separation distance is:

$$\mu_{BT} = \frac{(x^U - x^L)(y^U - y^L)}{4} \quad (4.19a)$$

- For a univariate concave term $UT(x)$, the maximum separation distance can be expressed as an convex optimisation problem so that μ_{UT} is given by:

$$\mu_{UT} = - \min_{x^L \leq x \leq x^U} \{-ut(x) + ut^L(x)\} \quad (4.19b)$$

where $ut^L(x)$ is the linearisation of $ut(x)$ around x^L .

- For a general nonconvex term, the maximum separation distance μ_{NT} is:

$$\mu_{NT} = \frac{1}{4} \sum_{i=1}^{n_z} \alpha_i (z_i^U - z_i^L)^2 \quad (4.19c)$$

where the α_i 's are defined by equation (4.9).

Given a node to be partitioned, the values of μ_{BT} , μ_{UT} and μ_{NT} are calculated for each term in accordance with its type. The term which appears to have the worst underestimator, e.g., the largest μ , is then used as a basis for the selection of the branching variables. Out of the set of the variables that participate in that term, the one with the *least reduced axis*, as defined by equation (4.18), is chosen for k -section. Under this strategy, the influence of the variable bounds on the quality of the underestimators is directly taken into account, and hence this adaptive branching scheme ensures the effective tightening of the lower bounding problem from iteration to iteration.

Strategy 3

This strategy is a variant of Strategy 2. Instead of computing the maximum separation distance between a term and its underestimator, their separation distance at the *optimum solution* of the lower bounding problem is used (Adjiman et al., 1998a):

- The measures μ_{BT} , μ_{TT} , μ_{FT} and μ_{FTT} for bilinear, trilinear, fractional and fractional trilinear terms, are:

$$\mu_{BT} = |x^* y^* - w_{BT}^*| \quad (4.20a)$$

$$\mu_{TT} = |x^* y^* z^* - w_{TT}^*| \quad (4.20b)$$

$$\mu_{FT} = \left| \frac{x^*}{y^*} - w_{FT}^* \right| \quad (4.20c)$$

$$\mu_{FTT} = \left| \frac{x^* y^*}{z^*} - w_{FTT}^* \right| \quad (4.20d)$$

where w_{BT} , w_{TT} , w_{FT} and w_{FTT} are the variables that are substituted for the bilinear term xy , trilinear term xyz , fractional term $\frac{x}{y}$ and fractional trilinear term $\frac{xy}{z}$ in order to construct their convex envelopes and the * superscript denotes the value of the variable at the solution of the current lower bounding problem.

- The measure for univariate concave term $ut(x)$ is:

$$\mu_{UT} = ut(x^*) - ut^L(x^*) \quad (4.20e)$$

- The general nonconvex term measure is:

$$\mu_{NT} = - \sum_{i=1}^{n_z} \alpha_i (z_i^U - z_i^*) (z_i^* - z_i^L) \quad (4.20f)$$

Strategy 4

The fourth branching procedure takes the approach of Strategy 3 one step further by considering the overall influence of each variable on the convex problem (Adjiman et al., 1998a). After the relevant measures μ_{BT} , μ_{TT} , μ_{FT} , μ_{FTT} , μ_{UT} and μ_{NT} have been calculated for every term, a measure μ of each variable's contribution may be obtained as follows:

$$\mu_i = \sum_{j \in BT_i} \mu_{BT}^j + \sum_{j \in TT_i} \mu_{TT}^j + \sum_{j \in FT_i} \mu_{FT}^j + \sum_{j \in FTT_i} \mu_{FTT}^j + \sum_{j \in UT_i} \mu_{UT}^j + \sum_{j \in NT_i} \mu_{NT}^j \quad (4.21)$$

where μ_i is the measure of the i -th variable. BT_i is the index set of the bilinear terms in which the i -th variable participates and μ_{BT}^j is the measure of the j -th bilinear term. TT_i is the index set of the trilinear terms in which the i -th variable participates and μ_{TT}^j is the measure of the j -th trilinear term. FT_i is the index set of the fractional terms in which the i -th variable participates and μ_{FT}^j is the measure of the j -th fractional term. FTT_i is the index set of the fractional trilinear terms in which the i -th variable participates and μ_{FTT}^j is the measure of the j -th fractional trilinear term. UT_i is the index set of univariate concave terms in which the i -th variable participates and μ_{UT}^j is the measure of the j -th univariate concave term. NT_i is the index set of the general nonconvex terms in which the i -th variable participates and μ_{NT}^j is the measure of the j -th general nonconvex term. The variable with the largest measure μ is selected as the branching variable, and k -section can be performed on it. If two or more variables have the same measure, the *least reduced axis* test is performed to distinguish them.

Note that branching strategies 2, 3, and 4 are particularly efficient since they take into account the sensitivity of the underestimators to the bounds used for each variable.

4.2.6 Variable Bound Updates

The quality of the convex lower bounding problem can also be improved by ensuring that the variable bounds are as tight as possible. In the current implementation of the α BB

algorithm, variable bound updates can either be performed at the onset of an α BB run or at each iteration.

In both cases, the same procedure is followed in order to construct the bound update problem. Given a solution domain, the convex underestimator for every constraint in the original NLP is formulated. The *bound problems* for variable z_i are then expressed as:

$$z_i^{L,\text{new}} = \begin{cases} \min_{\mathbf{z}} z_i \\ \text{s.t.} \\ \mathcal{L}_g(\mathbf{z}) \leq \mathbf{0} \\ \mathcal{L}_{\mathcal{J}}(\mathbf{z}) \leq UBD \\ \mathbf{z}^L \leq \mathbf{z} \leq \mathbf{z}^U \end{cases} \quad (4.22a)$$

$$z_i^{U,\text{new}} = \begin{cases} \min_{\mathbf{z}} -z_i \\ \text{s.t.} \\ \mathcal{L}_g(\mathbf{z}) \leq \mathbf{0} \\ \mathcal{L}_{\mathcal{J}}(\mathbf{z}) \leq UBD \\ \mathbf{z}^L \leq \mathbf{z} \leq \mathbf{z}^U \end{cases} \quad (4.22b)$$

where $\mathcal{L}_{\mathcal{J}}(\mathbf{z})$ and $\mathcal{L}_g(\mathbf{z})$ are the convex underestimators of the objective function and of the constraints respectively, \mathbf{z}^L and \mathbf{z}^U are the best calculated bounds on the variables, and UBD is the current best upper bound on the global solution. Once a new lower bound $z_i^{L,\text{new}}$ on z_i has been computed via problem (4.22a), this value is used in the formulation of problem (4.22b) for generation of an upper bound $z_i^{U,\text{new}}$.

Note that because of the computational expense incurred by an update of the bounds on all variables, it is often desirable to define a smaller subset of the variables on which this operation is to be performed. The criterion devised for the selection of the branching variables can be used in this instance since it provides a measure of the sensitivity of the problem to each variable. An option was therefore set up, in which bound updates are carried out only for a fraction of the variables with a non-zero μ , as calculated in equation (4.21).

5.1 Example 1

5.1.1 Problem Formulation

This model represents a first-order irreversible chain reaction $A \xrightarrow{k_1} B \xrightarrow{k_2} C$ as presented in Kaszonyi, private communication. Only the concentration of components A and B were measured, therefore, component C does not appear in the model used for estimation and its concentration can be simply calculate from substances A and B . The ODE model is of the form

$$\dot{x}_1 = -p_1 x_1 \quad (5.1a)$$

$$\dot{x}_2 = p_1 x_1 - p_2 x_2 \quad (5.1b)$$

$$\mathbf{x}_0 = [2, 0] \quad (5.1c)$$

where the state vector, \mathbf{x} , is defined as concentration of substances A and B $[c_A, c_B]$ and the parameter vector, \mathbf{p} , is defined as kinetic rate constants $[k_1, k_2]$. The data used in the study (see Table 5.1) were generated with values for the parameters of $\mathbf{p} = [0.8, 0.3]$ with no added error.

Finally, using the known values of the components A and B we can create an objective function for comparing the known data $\bar{\mathbf{x}} = [\bar{c}_A, \bar{c}_B]$ (see Table 5.1) at each point with the state variables in the model $\hat{\mathbf{x}} = [\hat{c}_A, \hat{c}_B]$. Let j be the index corresponding to each data

t [min]	c_A [mol/dm ³]	c_B [mol/dm ³]	t [min]	c_A [mol/dm ³]	c_B [mol/dm ³]
0	2.000	0.000	5.5	0.025	0.575
0.5	1.341	0.609	6	0.016	0.503
1	0.899	0.933	6.5	0.011	0.438
1.5	0.602	1.077	7	0.007	0.380
2	0.404	1.110	7.5	0.005	0.329
2.5	0.271	1.079	8	0.003	0.285
3	0.181	1.011	8.5	0.002	0.246
3.5	0.122	0.925	9	0.001	0.213
4	0.082	0.833	9.5	0.001	0.184
4.5	0.055	0.742	10	0.001	0.158
5	0.037	0.655			

Table 5.1: Example 1 – Experimental Data

point

$$\min_{\hat{\mathbf{x}}, \mathbf{p}} \sum_{i=1}^2 \sum_{j=1}^{10} (\hat{x}_i(t_j) - \bar{x}_i(t_j))^2 \quad (5.1d)$$

5.1.2 Results

DYNOPT used 5 collocation points for approximation of state profiles on 4 equal-length finite elements. The bounds on variables being set to $\mathbf{x}^{L,U} = [0 \ 2; 0 \ 2]$, $\mathbf{p}^{L,U} = [0 \ 1; 0 \ 1]$, in both solvers. Running optimization 100 times we get several optima sorted in Table 5.3 for DYNOPT and Table 5.4 for GDOC. As we can see, cost function values are spreaded almost identically. Also, the best obtained minima from both multistarts are equal to global solution presented in Table 5.2 with ϵ certainty. Each approach reports similar value of estimated parameters. Integration with found parameter values against original measurements are displayed in Figure 5.1 and Figure 5.2 where the best obtained values are used. The results fitted experimental values precisely.

Minimum	obj value	p_1	p_2
Global	0.00017604	0.8004760	0.2997822

Table 5.2: Example 1 – α BB method performed by GDOC

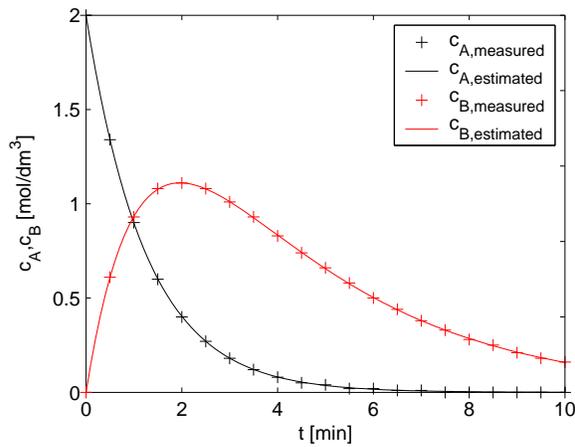


Figure 5.1: Example 1: DYNOPT

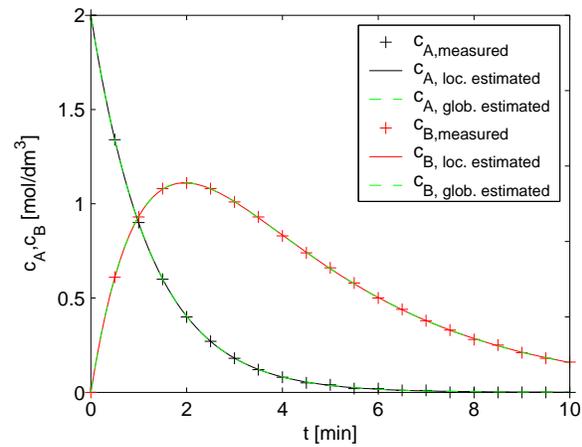


Figure 5.2: Example 1: GDOC

Minimum	obj value	p_1	p_2	frequency (%)
Local	0.00017604	0.8004062859	0.2997848632	96.4
Local	0.00017605	0.8004062276	0.2997850392	1.7
Local	0.00017606	0.8004101658	0.2997855893	0.9
Local	0.00017610	0.8004612658	0.2997856149	0.2
Local	0.00017612	0.8004058796	0.2997849440	0.1
Local	0.00017613	0.8004062276	0.2997850392	0.2
Local	0.00017614	0.8004612658	0.2997856149	0.2
Local	0.00017631	0.8004062859	0.2997848632	0.3

Table 5.3: Example 1 – Multistart method with 100 runs performed by DYNOPT (some picked values of objective function)

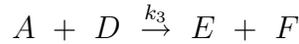
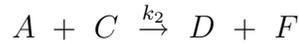
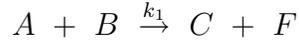
Minimum	obj value	p_1	p_2	frequency (%)
Local	0.00017605	0.8004947	0.2997755	76.5
Local	0.00017607	0.8004217	0.2997820	9
Local	0.00017608	0.8005090	0.2997815	5.4
Local	0.00017609	0.8004903	0.2997791	4.5
Local	0.00017610	0.8004947	0.2997755	2.6

Table 5.4: Example 1 – Multistart method with 100 runs performed by GDOC (some picked values of objective function)

5.2 Example 2

5.2.1 Problem formulation

This model is theoretical kinetic model described by reactions



published in Himmelblau (1970). Only the concentration of component A was measured.

The ODE model is of the form

$$\dot{x}_1 = -p_1x_1x_2 - p_2x_1x_3 - p_3x_1x_4 \quad (5.2a)$$

$$\dot{x}_2 = -p_1x_1x_2 \quad (5.2b)$$

$$\dot{x}_3 = p_1x_1x_2 - p_2x_1x_3 \quad (5.2c)$$

$$\dot{x}_4 = p_2x_1x_3 - p_3x_1x_4 \quad (5.2d)$$

$$\dot{x}_5 = p_3x_1x_4 \quad (5.2e)$$

$$\mathbf{x}_0 = [0.02090, 0.00697, 0, 0, 0] \quad (5.2f)$$

where the state vector, \mathbf{x} is defined as concentration of substances $[c_A, c_B, c_C, c_D, c_E]$ and the parameter vector, \mathbf{p} , is defined by kinetic rates constants $[k_1, k_2, k_3]$. The data used in the study are shown in Table 5.5.

Again, using the known values of the component A we can create an objective function for comparing the known data $\bar{\mathbf{x}} = [\bar{c}_A]$ (see Table 5.5) at each point with the state variables in the model $\hat{\mathbf{x}} = [\hat{c}_A]$. Let i be the index corresponding to each data point.

$$\min_{\hat{\mathbf{x}}, \mathbf{p}} \sum_{i=1}^{22} (\hat{x}(t_i) - \bar{x}(t_i))^2 \quad (5.2g)$$

5.2.2 Results

Dynopt used 8 collocation points for approximation of state profiles on 8 equal-length finite elements. Table 5.6 shows some local minima from 100 multistarts running DYNOPT. In Table 5.2 can be seen global minimum obtained by α BB method and some local minima from 100 multistarts running GDOC. DYNOPT results lower performance indices than GDOC (See Table 5.7). It is probably caused by difference in approaches. While in DYNOPT, both state and control profiles are discretized on each interval, in GDOC are

t [min]	$c_A \times 10^3$ [mol/liter]	t [min]	$c_A \times 10^3$ [mol/liter]
4.50	51.400	76.75	8.395
8.67	14.220	90.00	7.891
12.67	13.350	102.00	7.510
17.75	12.320	108.00	7.370
22.67	11.810	147.92	6.646
27.08	11.390	198.00	5.883
32.00	10.920	241.75	5.322
36.00	10.540	270.25	4.960
46.33	9.780	326.25	4.518
57.00	9.157	418.00	4.075
69.00	8.594	501.00	3.372

Table 5.5: Example 2 – Experimental Data

control profiles discretized and state profiles are integrated on each time interval. A small difference between predicted model and measured samples can be observed in both cases on pictures Figure 5.3, Figure 5.4 and in Table 5.6, Table 5.7 and Table 5.8. The best obtained values are used for comparisons.

Minimum	obj value	p_1	p_2	p_3	frequency (%)
Local	0.00401	22.84936	1.48237	0.30374	19
Local	0.00402	21.80278	1.48678	0.30331	18
Local	0.00403	24.28812	1.47666	0.30433	20
Local	0.00406	21.75700	1.48682	0.30329	10
Local	0.00408	22.03482	1.48611	0.30336	4
Local	0.00411	22.07459	1.48742	0.30317	1

Table 5.6: Example 2 – Multistart method with 100 runs performed by DYNOPT (some picked values of objective function)

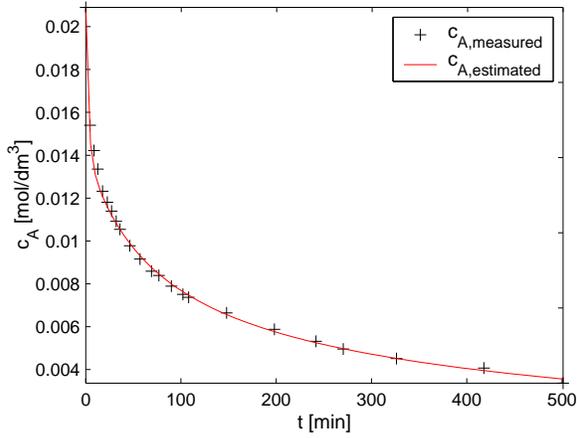


Figure 5.3: Example 2: DYNOPT

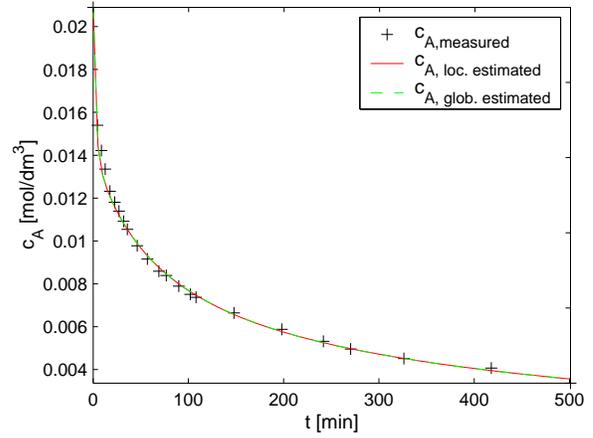


Figure 5.4: Example 2: GDOC

Minimum	obj value	p_1	p_2	p_3	frequency (%)
Local	0.00469	23.65910	1.48098	0.30351	24
Local	0.00470	23.00078	1.48313	0.30371	49
Local	0.00471	19.51164	1.49981	0.30188	1
Local	0.00472	23.68011	1.47900	0.30418	1
Local	0.00473	26.11833	1.47158	0.30474	2
Local	0.00474	23.00078	1.48313	0.30371	3
Local	0.00475	23.90928	1.47906	0.30394	1

Table 5.7: Example 2 – Multistart method with 100 runs performed by GDOC (some picked values of objective function)

Minimum	obj value	p_1	p_2
Global	0.00468	23.63440	1.48019

Table 5.8: Example 2 – α BB method performed by GDOC

5.3 Example 3

5.3.1 Problem formulation

This model is a theoretical kinetic model published in Himmelblau (1970). The concentration of components A and B were measured. The ODE model is of the form

$$\dot{x}_1 = -p_1x_1x_2 - p_2x_1x_4 \quad (5.3a)$$

$$\dot{x}_2 = -p_1x_1x_2 \quad (5.3b)$$

$$\dot{x}_3 = p_1x_1x_2 \quad (5.3c)$$

$$\dot{x}_4 = p_1x_1x_2 + p_2x_1x_4 \quad (5.3d)$$

$$\dot{x}_5 = p_2\sqrt{x_1x_4} \quad (5.3e)$$

$$\mathbf{x}_0 = [1.5, 1, 0, 0, 0] \quad (5.3f)$$

where the state vector, \mathbf{x} is defined as concentration of substances $[c_A, c_B, c_C, c_D, c_E]$ and the parameter vector, \mathbf{p} , is defined by kinetic rate constants $[k_1, k_2]$. The data used in the study (see Table 5.9) were generated with values for the parameters of $\mathbf{p} = [0.3, 0.1]$ with added small error.

Again, using the known values of the components A and B we can create an objective function for comparing the known data $\bar{\mathbf{x}} = [\bar{c}_A, \bar{c}_B]$ (see Table 5.9) at each point with the state variables in the model $\hat{\mathbf{x}} = [\hat{c}_A, \hat{c}_B]$. Let j be the index corresponding to each data point.

$$\min_{\hat{\mathbf{x}}, \mathbf{p}} \sum_{i=1}^2 \sum_{j=1}^{10} (\hat{x}_i(t_j) - \bar{x}_i(t_j))^2 \quad (5.3g)$$

5.3.2 Results

In this case study, Table 5.10 provides same results from 100 different starting points for parameters using DYNOPT. GDOC refers twice lower value of cost functions in Table 5.11 and global solution displayed in Table 5.12. In this case is clear that CVP method is more precise compared to TP. From Figure 5.5 and Figure 5.6 is visible that quality of parameter values obtained by GDOC are more precise than values obtained by DYNOPT. The best obtained values are used for comparisons. It seems that with small number of measured data, which are also dispersed, DYNOPT could not fit very well. It is caused by oscilation of Lagrange polynomial in TP against smooth integrated curved provided by CVP method. DYNOPT works with 4 same interval's length and 4 colocation points.

t [min]	c_A [mol/liter]	c_B [mol/liter]
1	1.1529	0.6747
2	0.9333	0.4944
3	0.7806	0.3828
4	0.6675	0.3083
5	0.5801	0.2558
6	0.5104	0.2173
7	0.4535	0.1881
8	0.4060	0.1653
9	0.3658	0.1473
10	0.3314	0.1327

Table 5.9: Example 3 – Experimental Data

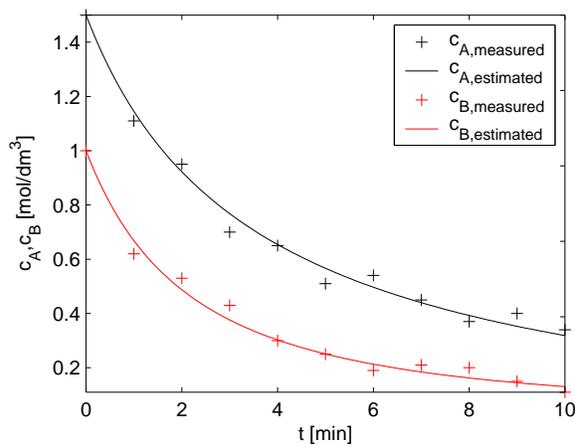


Figure 5.5: Example 3: DYNOPT

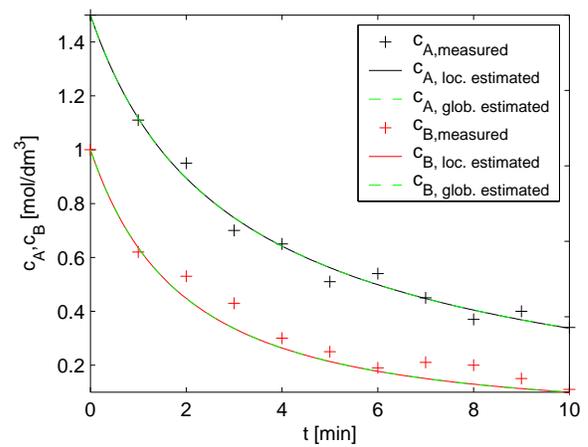


Figure 5.6: Example 3: GDOC

Minimum	obj value	p_1	p_2	frequency (%)
Local	0.02595	0.30804	0.10591	100

Table 5.10: Example 3 – Multistart method with 100 runs performed by DYNOPT

Minimum	obj value	p_1	p_2	frequency (%)
Local	0.01206579	0.3514424	0.07976015	31.1
Local	0.01206580	0.3514051	0.07977576	38.9
Local	0.01206581	0.3514051	0.07977576	13
Local	0.01206582	0.3514051	0.07977576	5.7
Local	0.01206583	0.3514274	0.07980991	5.9
Local	0.01206584	0.3514051	0.07977576	4

Table 5.11: Example 3 – Multistart method with 100 runs performed by GDOC (some picked values of objective function)

Minimum	obj value	p_1	p_2
Global	0.01206578	0.3514370	0.07976381

Table 5.12: Example 3 – α BB method performed by GDOC

5.4 Example 4

5.4.1 Problem formulation

This model represents the cyclohexadienyl radical reaction with oxygen in non-polar solvents as presented in Taylor et al. (2004). Only the absorbance for cyclohexadienyl radicals was measured. The ODE model is of the form

$$\begin{aligned} \dot{x}_1 = & q_1 x_2 x_3 - q_3(p_1 + p_2)x_1 + (p_1/q_4)x_4 + \\ & + (p_2/q_5)x_5 - q_2 x_1^2 \end{aligned} \quad (5.4a)$$

$$\dot{x}_2 = -q_1 x_2 x_3 \quad (5.4b)$$

$$\dot{x}_3 = -q_1 x_2 x_3 \quad (5.4c)$$

$$\dot{x}_4 = p_1 x_1 q_3 - (p_1/q_4)x_4 \quad (5.4d)$$

$$\dot{x}_5 = p_2 q_3 x_1 - (p_3 + p_2/q_5)x_5 \quad (5.4e)$$

$$\mathbf{x} = [0, 0.00014, 0.4, 0, 0] \quad (5.4f)$$

$$\mathbf{q} = [53, 1200, 0.0019, 2081, 4162] \quad (5.4g)$$

where the state vector, \mathbf{x} , is defined as [*c*-C₆H₇, (CH₃)₃CO, 1, 4-C₆H₈, *p*-C₆H₇OO, *o*-C₆H₇OO], and the parameter vector, \mathbf{p} , is defined as [*k*₂, *k*₃, *k*₄] (see (Taylor et al., 2004), (Singer et al., 2006)).

Finally, using the known values of the absorbance $\bar{\mathbf{d}}$ for cyclohexadienyl radical we can create an objective function. The known values of absorbance \bar{d}_i at each point are compared with calculated absorbance values from the model. The state vector is defined as $\hat{\mathbf{x}} = [c_A, c_B, c_C, c_D, c_E]$ and then predicted absorbance $\hat{\mathbf{d}}$ is equal to $2100\hat{x}_1(t_i) + 200(\hat{x}_4(t_i) + \hat{x}_5(t_i))$. Let *i* be the index corresponding to each data point. Cost function is described as

$$\min_{\hat{\mathbf{x}}, \mathbf{p}} \sum_{i=1}^{460} (\hat{d}(t_i) - \bar{d}(t_i))^2 \quad (5.4h)$$

5.4.2 Results

Using 10 collocation points for approximation of state profiles on 11 finite elements of lengths [0.46; 4], with bounds on variables being set to, $\mathbf{p}^{L,U} = [10 \ 1200; 10 \ 1200; 0.001 \ 40]$ we obtain results printed in Table 5.13. Comparison between the integrated system with estimated parameters values and the original data is shown on Figure 5.7 where the best found local optimum is used. GDOC did not converge.

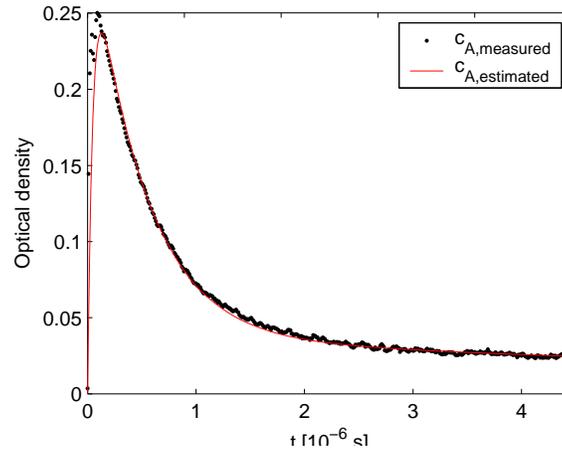


Figure 5.7: Example 4: DYNOPT

Minimum	obj value	p_1	p_2	p_3	frequency (%)
Local	0.03916	531.2500	400.7325	32.0750	1
Local	0.04014	550.5497	1084.2537	0.2716	1
Local	0.06380	29.7843	978.0523	19.4768	1
Local	0.10122	662.4405	1114.4850	0.2796	1
Local	0.14415	1146.5787	374.6621	24.1901	1
Local	0.52467	1081.2793	401.4550	24.5058	1

Table 5.13: Example 4 – Multistart method with 100 runs performed by DYNOPT (some picked values of objective function)

Conclusions

Parameter estimation becomes an important part of chemical computing in matter of determining of parameters in semi-known models that describe variety of chemical technologies. Major difficulty is to locate almost true values from noisy data obtained from experimental measurements. To express relationship between measurements and appropriate model data, Chapter 2 defined objective function based on error-on-variables approach that maximize likelihood of occurrence certain parameter values in all experimental data. In other words, we minimize weighted least-squared error between observations and predictions.

Bigger precision is needed in model described with differential algebraic equations or ordinary differential equations for parameter estimation problems. In Chapter 4 we introduced two numerical methods and we compared them. One which integrates original process equations and second which does not integrate, but discretizes and approximates states by Lagrange polynomial with Legendre roots. In both methods control vector is provided by measured samples. This method, called orthogonal collocation, transfers dynamic system into static nonlinear problem (NLP) which is then solved with nonlinear solvers. As it is obvious, the second method is less precise than the first one.

All previous steps were sufficient for locally optimal solutions. A local solution depends on an initial set of parameters. To improve precision of results we test two methods on several examples presented in Chapter 4. Multistart is a very simple method and as name says it is based on selected count of local optimizations with randomly generated initial set of optimized parameters. On the other side, αBB method searches through intervals

that are branched into smaller ones. At each iteration we also compute α coefficient from Hessians of original functions and by underestimating ensure that the solution is really global. Using CVP method we are unable to compute exact α s, because it does not operate with Hessians. Advantage of orthogonal collocation is that we are able to compute α s at each iteration cycle. This can not be compared against CVP method because it is subject of further work.

There was another recent development beside improving quality of parameter estimation of DYNOPT. An user-friendly graphical interface and automatic gradient generation has been implemented. The problem inputting into DYNOPT is much more easier as is shown in Appendix A.

Bibliography

- C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, α BB, for general twice-differentiable constrained NLPs - II. Implementation and computational results. *Computers and Chemical Engineering*, 22(9):1159–1179, 1998a.
- C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A global optimization method, α BB, for general twice-differentiable constrained NLPs - I. Theoretical advances. *Computers and Chemical Engineering*, 22(9):1137–1158, 1998b.
- T. F. Anderson, D. S. Abrams, and E. A. Grens II. Evaluation of parameters for nonlinear thermodynamic models. *AIChE Journal*, (24):20, 1975.
- I. P. Androulakis, C. D. Maranas, and C. A. Floudas. α BB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995.
- Y. Bard. *Nonlinear Parameter Estimation*. Academic Press, New York, 1974.
- Y. Bard and L. Lapidus. Kinetics analysis by digital parameter estimation. *Catalysis Reviews*, (2):67, 1968.
- R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- R. Bellman, J. Jacquez, R. Kalaba, and S. Schwimmer. Quasilinearization and estimation of chemical rate constraints from raw kinetic data. *Math Biosci*, (1):71, 1967.
- H. I. Britt and R. H. Luecke. The estimation of parameters in nonlinear implicit models. *Technometrics*, (15):233, 1973.

- A. E. Bryson and Y. Ho. *Applied Optimal Control*. Hemisphere Publishing Corporation, Washington, D.C., 1975.
- C. T. Chen and C. Hwang. Optimal control computation for differential-algebraic process systems with general constraints. *Chem. Eng. Comm.*, (97):9–26, 1990.
- J. E. Cuthrell and L. T. Biegler. On the optimization of differential-algebraic process systems. *AIChE Journal*, 33:1257–1270, 1987.
- J. E. Cuthrell and L. T. Biegler. Simultaneous optimization and solution methods for batch reactor control profiles. *Computers and Chemical Engineering*, 13(1/2):49–62, 1989.
- W. R. Esposito and C. A. Floudas. Global optimization for the parameter estimation of differential-algebraic systems. *Ind. Eng. Chem. Res.*, (39):1291–1310, 2000.
- W. F. Feehery. *Dynamic Optimisation with Path Constraints*. PhD thesis, MIT, 1998.
- M. Fikar and M. A. Latifi. User’s guide for FORTRAN dynamic optimisation code DYNO. Technical Report mf0201, LSGC CNRS, Nancy, France; STU Bratislava, Slovak Republic, 2002.
- C. A. Floudas. *Deterministic Global Optimization: Theory, methods and applications. Nonconvex optimization and its applications*. Kluwer Academic Publishers, 2000.
- C. J. Goh and K. L. Teo. Control parametrization: a unified approach to optimal control problems with general constraints. *Automatica*, (10):3–18, 1988a.
- C. J. Goh and K. L. Teo. Control parametrization: a unified approach to optimal control problems with general constraints. *Automatica*, (10):3–18, 1988b.
- D. M. Himmelblau. *Process Analysis by Statistical Methods*. John Wiley and Sons Inc, New York, 1970.
- M. Hwang and J. H. Seinfeld. A new algorithm for the estimation of parameters in ordinary differential equations. *AIChE J.*, (18):90, 1972.
- D. Jacobson and M. Lele. A transformation technique for optimal control problems with a state variable inequality constraint. *IEEE Trans. Automatic Control*, 5:457–464, 1969.
- N. Kalogerakis and R. Luus. Simplification of quasilinearization method for parameter estimation. *AIChE J.*, (29):858, 1983.

- A. Kaszonyi. *private communication*.
- M. J. Liebman, T. F. Edgar, and L. S. Lasdon. Efficient data reconciliation and estimation for dynamic processes using nonlinear programming techniques. *Comp. Chem. Eng.*, (16):963, 1992.
- J. S. Logsdon and L. T. Biegler. Accurate solution of differential-algebraic optimization problems. *Chemical Engineering Science*, (28):1628–1639, 1989.
- J. S. Logsdon and L. T. Biegler. Decomposition strategies for large-scale dynamic optimization problems. *Chemical Engineering Science*, 47(4):851–864, 1992.
- R. Luus. Parameter estimation of lotka-volterra problem by direct search optimization. *Hung. J Ind. Chem.*, (26):287, 1998.
- C. D. Maranas and C. A. Floudas. Global optimization in generalized geometric programming. *Computers and Chemical Engineering*, 21(4):351–369, 1997.
- The MathWorks. *Symbolic Math Toolbox for use with MATLAB: User's Guide*, 1998.
- The MathWorks. *Optimization Toolbox for use with MATLAB: User's Guide*, 2006.
- L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes*. Pergamon Press, New York, 1964.
- H. Schwetlick and V. Tiller. Numerical methods for estimating parameters in nonlinear models with error in the variables. *Technometrics*, (27):17, 1985.
- A. B. Singer, Benoit Chachuat, and Barton P. I. *GDOC manual*. Department of Chemical Engineering Massachusetts Institute of Technology, Massachusetts, 2005.
- A. B. Singer, J. W. Taylor, P. I. Barton, and W. H. Green. Global dynamic optimization for parameter estimation in chemical kinetics. 110:971–976, 2006.
- W. H. Southwell. Fitting experimental data. *J. of Comp. Physics*, (4):465, 1969.
- J. W. Taylor, G. Ehlker, H-H. Carstensen, L. Rulsen, R. W. Field, and W. H. Green. Direct measurement of the fast, reversible reaction of cyclohexadienyl radicals with oxygen in nonpolar solvents. *Journal of Physical Chemistry A*, 108(35):7193–7203, 2004.
- T. B. Tjoa and L. T. Biegler. Simultaneous solution and optimization strategies for parameter estimation of differential algebraic equation systems. *Ind. Eng. Chem. Res.*, (30):376, 1991.

- B Van Den Brosch and L. A. Hellinckx. A new method for the estimation of parameters in differential equations. *AICHE J.*, (20):250, 1974.
- M. Čižniar, M. Fikar, and M. A. Latifi. *MATLAB Dynamic Optimisation Code DYNOPT. User's Guide, version 4.1.0*. KIRP FCHPT STU, Bratislava, 2006.
- M. Čižniar, M. Podmajersky, M. Fikar, and M. A. Latifi. *MATLAB Dynamic Optimisation Code DYNOPT. User's Guide, version 4.1.1*. KIRP FCHPT STU, Bratislava, 2007.
- M. Čižniar, D. Salhi, M. Fikar, and M. A. Latifi. A MATLAB package for orthogonal collocations on finite elements in dynamic optimisation. In *Proc. 15. Int. Conference Process Control '05*, page 058f.pdf, Štrbské Pleso, High Tatras, Slovakia, 2005.
- J. Villadsen and M. L. Michelsen. *Solution of Differential Equation Models by Polynomial Approximation*. Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1978.
- A. Víteček and M. Vítečková. *Optimální Systémy Řízení (Optimal Control Systems)*. VŠT–Technická Univerzita Ostrava, 2002.

Dynopt GUI and Symdynopt

Since the first version of DYNOPT package was released (Čižniar et al., 2005), there was continuing development process (see <http://www.kirp.chnikf.stuba.sk/~fekar>) and at present, the package earns improved computational effort: the problems with varying levels of difficulty such as minimum time problems, unconstrained problems and constrained problems applied at the final time or over the full time interval, and parameter estimation problems by simple start or using multistart option, is able to solve.

All the aforementioned features did not refer to user's comfort due to expected deeper knowledge and practise with the derivatives. This can affect the computational procedure in a way of generating and providing gradients manually. From user's point of view, to define the gradients for a given complex problem can be slow and painful. On the other hand, the evaluation of such problems without provided gradients, is almost impossible.

The aim of this package is to be as much as possible user-friendly, user should no more think about the gradients. The tool *symdynopt* for generating the analytical gradients automatically instead of being generated by user manually is developed and implemented within DYNOPT package in the *dynoptfunctions* function. Moreover, to make the definition of the optimization problem more comfortable, the graphical interface *dynoptgui* has been developed, too.

The generation of analytical gradients is based on functions from Matlab Symbolic Toolbox. The closer look and differences between previous and current inputs are demonstrated in Sections A.2 and A.3 as recent developments in DYNOPT package.

A.1 Example definition

Consider the following problem (Feehery, 1998; Fikar and Latifi, 2002; Jacobson and Lele, 1969)

$$\min_{\mathbf{u}(t)} J = \int_0^1 (x_1^2 + x_2^2 + 0.005u^2)dt \quad (\text{A.1a})$$

subject to the process described by a set of 2 ODEs:

$$\dot{x}_1 = x_2, \quad x_1(0) = 0 \quad (\text{A.1b})$$

$$\dot{x}_2 = -x_2 + u, \quad x_2(0) = -1 \quad (\text{A.1c})$$

such that following state path constraint needs to be satisfied:

$$x_2 - 8(t - 0.5)^2 + 0.5 \leq 0, \quad t \in [0, 1] \quad (\text{A.1d})$$

with terminal time $t_f = 1$. $x_1(t)$, $x_2(t)$ as states and $u(t)$ as control variable. As DYNOPT needs the objective function in the Mayer form, an additional differential equation will be defined

$$\dot{x}_3 = x_1^2 + x_2^2 + 0.005u^2, \quad x_3(0) = 0 \quad (\text{A.1e})$$

and therefore, the cost (A.1a) will be rewritten to

$$\min_{\mathbf{u}(t)} J = x_3(t_f) \quad (\text{A.1f})$$

A.2 Symdynopt

Dynopt function requests tree input files: *process*, *objfun*, *confun*. These functions contain necessary values for further optimization divided into flags. The *dynoptfunctions* function fill them with appropriate data, constructed from user's input. There were two different ways that simplify the problem formulation: (i) to add another layer, which builds requested files from modified user's input, or (ii) to avoid algorithm from input output operations and to evaluate all equations in memory. To keep previous syntax and to elude of complications with new problem definition or brand new file syntax, *symdynopt* is based on Matlab Symbolic Toolbox (MathWorks, 1998) as a stand-alone application included in DYNOPT bundle. It does not interfere with original *dynopt* code and can be used with all DYNOPT 4.x.x versions.

Let's describe how the procedure works. Input into *dynoptfunctions* function is as follows: number of states, controls and parameters variables, and also main equations. The input equations must have string form because Symbolic toolbox operates with symbolic

objects and then conversion between them is simpler. In original *dynopt* m-functions are equations of NLP formulation and appropriate partial derivatives. In cycle, step-by-step, *dynopt* m-functions are filled with process ODEs and their partial derivatives by all state, control and parameter vectors. These vectors are built in compliance with given number of states, controls and parameters. Then are used in *Jacobian*, as it is demonstrated below, to generate symbolic matrix of partial derivatives. In next step, this matrix is converted into string form, which satisfies *dynopt* m-function's syntax and can be inserted into it.

Next, we show how Matlab's function *Jacobian* is used. At first, database of symbolic objects is created

```
>> syms x1 x2 x3 u
```

Next, string description of equation is converted using another Matlab's functions *fprintf* and *eval* into symbolic object

```
>> f = x1^2 + x2^2 + 0.005*u^2
f =
x1^2+x2^2+1/200*u^2
```

No we are able to utilise built-in routines of Symbolic toolbox. The partial derivatives by vector \mathbf{x} and \mathbf{u} are given as

```
>> dfdx = jacobian(f,[x1 x2 x3])
dfdx =
[ 2*x1, 2*x2,    0]
```

```
>> dfdu = jacobian(f,[u])
dfdu =
1/100*u
```

The complete usage of *symdynopt* tool is demonstrated and commented below.

In the previous versions of DYNOPT package, user had to define problem A.1 with the tree mentioned functions and fill them with many information as shown in steps 1–3:

Step1: Process

```
function sys = process(t,x,flag,u,p)

switch flag,
    case 0 % right sides of ODE/DAEs
```

```

        sys = [x(2);
              -x(2)+u;
              x(1)^2+x(2)^2+0.005*u^2];
case 1 % df/dx
        sys = [0 0 2*x(1);
              1 -1 2*x(2);
              0 0 0];
case 2 % df/du
        sys = [0 1 0.01*u];
case 3 % df/dp
        sys = [];
case 4 % df/dt
        sys = []; % [0 0 0]
case 5 % x0
        sys = [0;-1;0];
case 6 % dx0/dp
        sys = [];
case 7 % mass matrix M
        sys = [];
case 8 % unused flag, don't use it !
        sys = [];
otherwise
        error(['unhandled flag = ', ...
              num2str(flag)]);
end

```

Step2: Objective Function

```

function [f,Df] = objfun(t,x,u,p)

f = [x(3)];
Df.t = [];
Df.x = [0;0;1];
Df.u = [];
Df.p = [];

```

Step3: Constraints

```
function [c,ceq,Dc,Dceq] = ...
    confun(t,x,flag,u,p)

switch flag
    case 0 % constraints at t0
        c = [];
        ceq = [];

        % gradient calculus
        if nargout == 4
            Dc.t = [];
            Dc.x = [];
            Dc.u = [];
            Dc.p = [];
            Dceq.t = [];
            Dceq.x = [];
            Dceq.u = [];
            Dceq.p = [];
        end
    case 1 % constraints in [t0,tf]
        c = [x(2)-8*(t-0.5)^2+0.5];
        ceq = [];

        % gradient calculus
        if nargout == 4
            Dc.t = [-16*t+8];
            Dc.x = [0;1;0];
            Dc.u = [];
            Dc.p = [];
            Dceq.t = [];
            Dceq.x = [];
            Dceq.u = [];
            Dceq.p = [];
        end
end
```

```
case 2 % constraints at tf
    c = [];
    ceq = [];

    % gradient calculus
    if nargout == 4
        Dc.t = [];
        Dc.x = [];
        Dc.u = [];
        Dc.p = [];
        Dceq.t = [];
        Dceq.x = [];
        Dceq.u = [];
        Dceq.p = [];
    end
end
```

As it was mentioned, the aim was to make the problem definition user-friendly. This led to development of an additional tool, *symdynopt*. User can use function *dynoptfunctions* to construct the aforementioned functions, as it is demonstrated in the following code:

Step1-3: Using *symdynopt* Tool

```
eq.nx = 3;
eq.nu = 1;
eq.np = 0;
eq.M = [];

eq.shortcuts = [];
eq.process = {'x(2)';
             '-x(2)+u';
             'x(1)^2+x(2)^2+0.005*u^2'};
eq.x0 = {'0'; '-1'; '0'};
eq.objfun = {'x(3)'};
eq.confun.c.t0 = [];
eq.confun.ceq.t0 = [];
eq.confun.c.t0tf = {'x(2)-8*(t-0.5)^2+0.5'};
eq.confun.ceq.t0tf = [];
```

```
eq.confun.c.tf = [];  
eq.confun.ceq.tf = [];  
  
dynoptfunctions(eq)
```

Code can be run separately before or in front of, as a part of main optimization code. It will automatically generate the *process*, *objfun*, *confun* functions and fill them with all necessary informations shown in steps 1–3. It is done by exploiting the Matlab Symbolic Toolbox (MathWorks, 1998).

The rest of the procedure (i.e., optimization and interpretation of the results) stays unchanged.

Step4: Optimization

```
opt = optimset('LargeScale','off', ...  
              'Display','iter');  
opt = optimset(opt,'GradObj', 'on', ...  
              'GradConstr','on');  
opt = optimset(opt,'TolFun',1e-7);  
opt = optimset(opt,'TolCon',1e-7);  
opt = optimset(opt,'TolX',1e-7);  
  
optimparam.optvar = 3;  
optimparam.objtype = [];  
optimparam.ncolx = 6;  
optimparam.ncolu = 2;  
optimparam.li = (1/6)*ones(6,1);  
optimparam.tf = 1;  
optimparam.ui = zeros(1,6);  
optimparam.par = [];  
optimparam.bdu = [];  
optimparam.bdx = [];  
optimparam.bdp = [];  
optimparam.objfun = @objfun;  
optimparam.confun = @confun;  
optimparam.process = @process;
```

```
optimparam.options = opt;  
  
[optimout,optimparam]=dynopt(optimparam)
```

After the problem has been defined by the above mentioned functions, user calls the *dynopt* functions *profiles* and *constraints* as follows:

Step5: *Interpretation of Results*

```
[tplot,uplot,xplot] = ...  
    profiles(optimout,optimparam,10);  
[tp,cp,ceqp] = ...  
    constraints(optimout,optimparam,10);
```

More information about DYNOPT package and *symdynopt* tool also is in User's Guide (Čižniar et al., 2007).

A.3 Dynopt GUI

A fully interactive graphical user interface has been developed for *dynopt*. This add-on depends on *symdynopt*, but does not affect original *dynopt* tool. Firstly, user passes through wizard, then necessary files will be generated and executed. The aim was to create a transparent interface, easy to use, to accelerate inputs, to report errors from checking fields and missing fields, and to display content evolving from previous selections.

A.3.1 Step 1: Create or Load a Problem

User begins with selection of a type of optimization problem (see Figure A.1).

A.3.2 Step 2: Optimization Options

Here maximal number of function evaluations, iterations and tolerance for objective function, constraints and variables, as well as a choice of optimization variables can be selected (see Figure A.2).

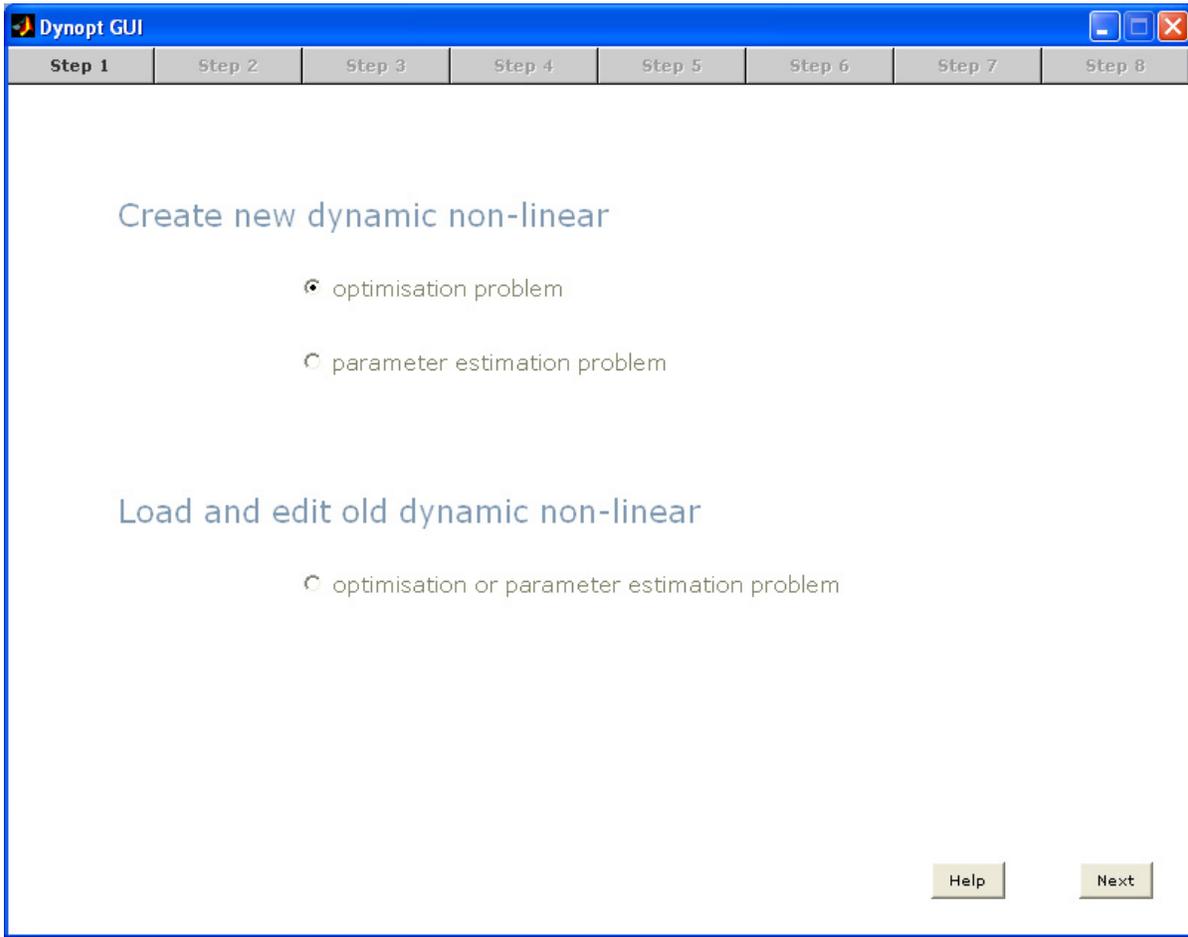


Figure A.1: Optimization type selection

A.3.3 Step 3: Initialisation Values for Optimization

Step 3 asks for number of state, control and parameter variables, dynamic model and constraints, number of collocation points for states and controls that describe quality of discretisation procedure, width of intervals, final time value, bounds to states, controls and parameters, path to measured data and number of multistart evaluations. Not every field is obligatory. The number of obligatory entries depends on records from previous steps. If certain inputs are unnecessary for given problem, the field is disabled. Provided data must pass the check of validity. If they do not pass, an error message appears to point at bad entries (see Figure A.4 and Figure A.3).

A.3.4 Step 4: Cost and Process

In tab **step 4** the cost function and process model have to be defined. It is done via a simple input dialog box shown Figure A.6. Process equations are stored in the listbox

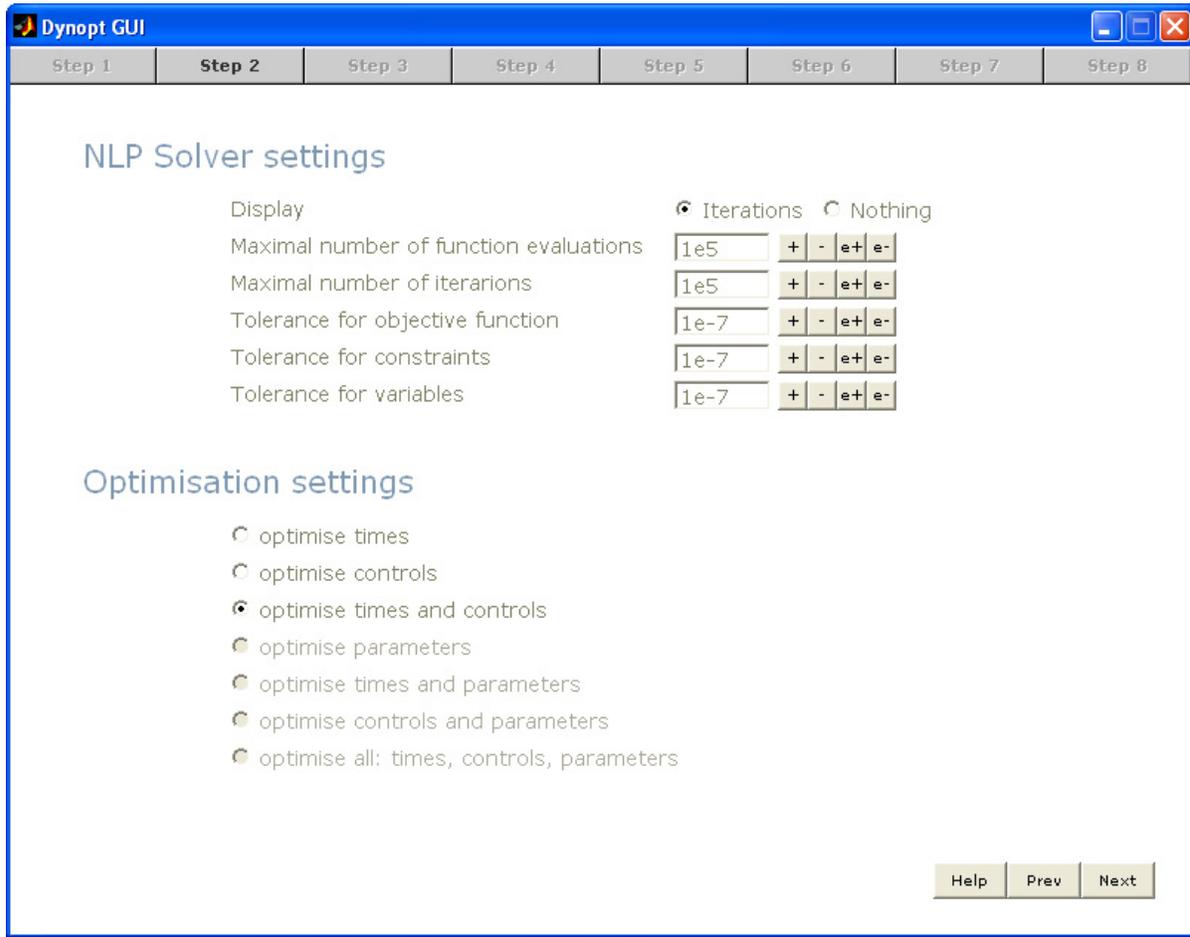


Figure A.2: Bundle of initial optimization settings

where they can be simply added, edited, positioned or cleared (see Figure A.5).

A.3.5 Steps 5-7: Constraints

Constraints can be entered very similar to previous tab. They are provided by listbox and appropriate input dialog (see Figure A.7, Figure A.8, and Figure A.9).

A.3.6 Step 8: Save and Solve Problem

The Last step is to save (in Figure A.11) the problem and run main optimization process (see Figure A.10). Then *symdynopt* generates derivations automatically and calls *dynopt* that solves the problem.

Dynopt GUI

Step 1 Step 2 **Step 3** Step 4 Step 5 Step 6 Step 7 Step 8

Initials and boundaries

Number of states:

Number of controls:

Number of parameters:

Number of collocation points for states:

Number of collocation points for controls:

Interval widths:

Bounds for states:

Bounds for controls:

Bounds for parameters:

Multistart:

Final time:

Path to experimental data:

Figure A.3: Form for definition of initials and bounds depends on preliminary selections

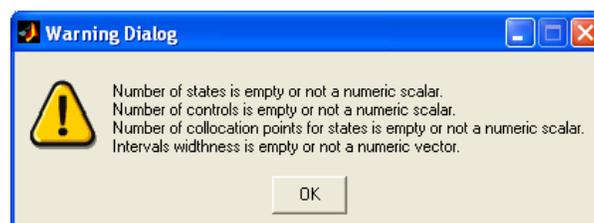


Figure A.4: Error message report for incorrect inputs

Dynopt GUI

Step 1 Step 2 Step 3 **Step 4** Step 5 Step 6 Step 7 Step 8

Objective function

Process

Initial conditions for states

Initial conditions for parameters

Mass matrix

Initial control values for each interval

ODE or DAE process description

Figure A.5: Form for cost function and process model

Process equations input

Enter process equation

Figure A.6: Equation input dialog

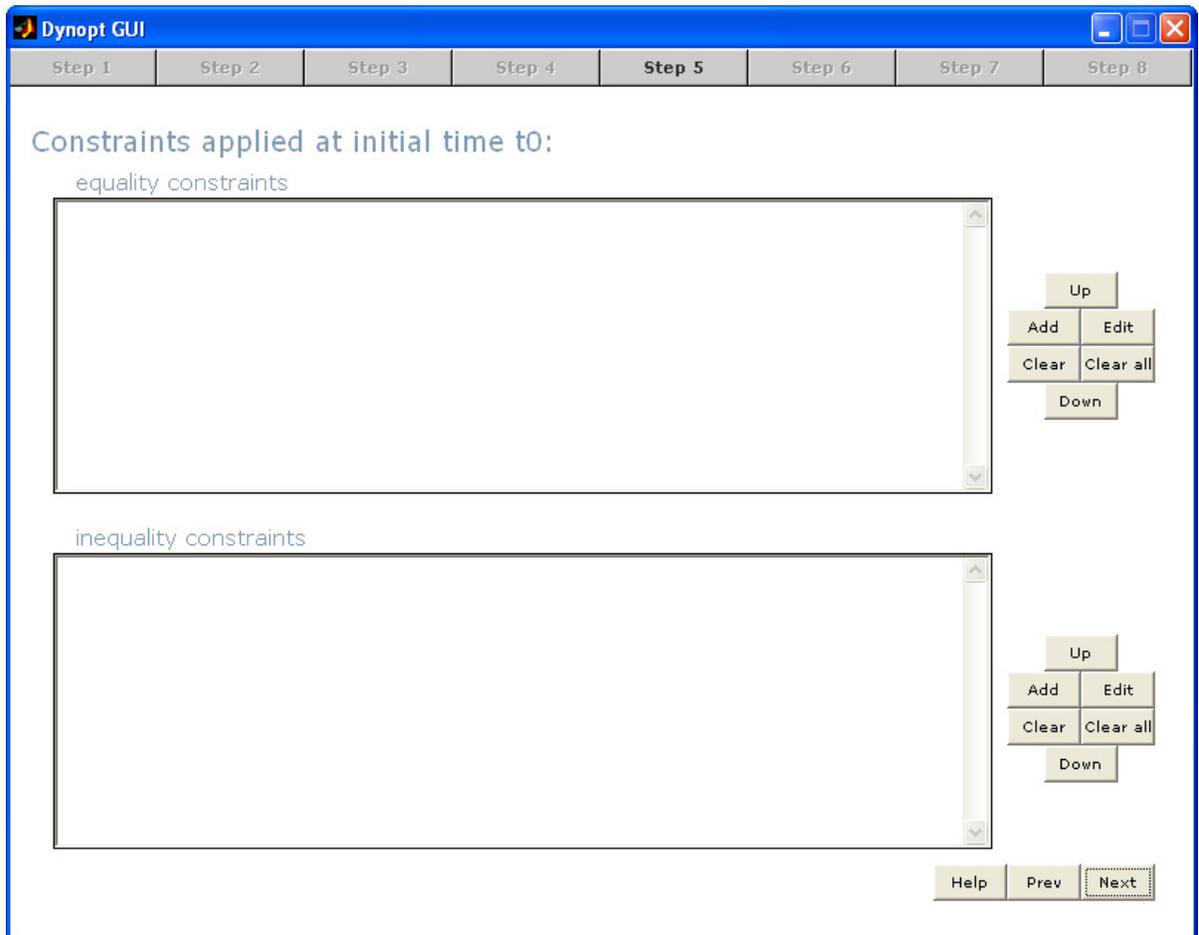


Figure A.7: Form for equality and inequality constraints at initial time

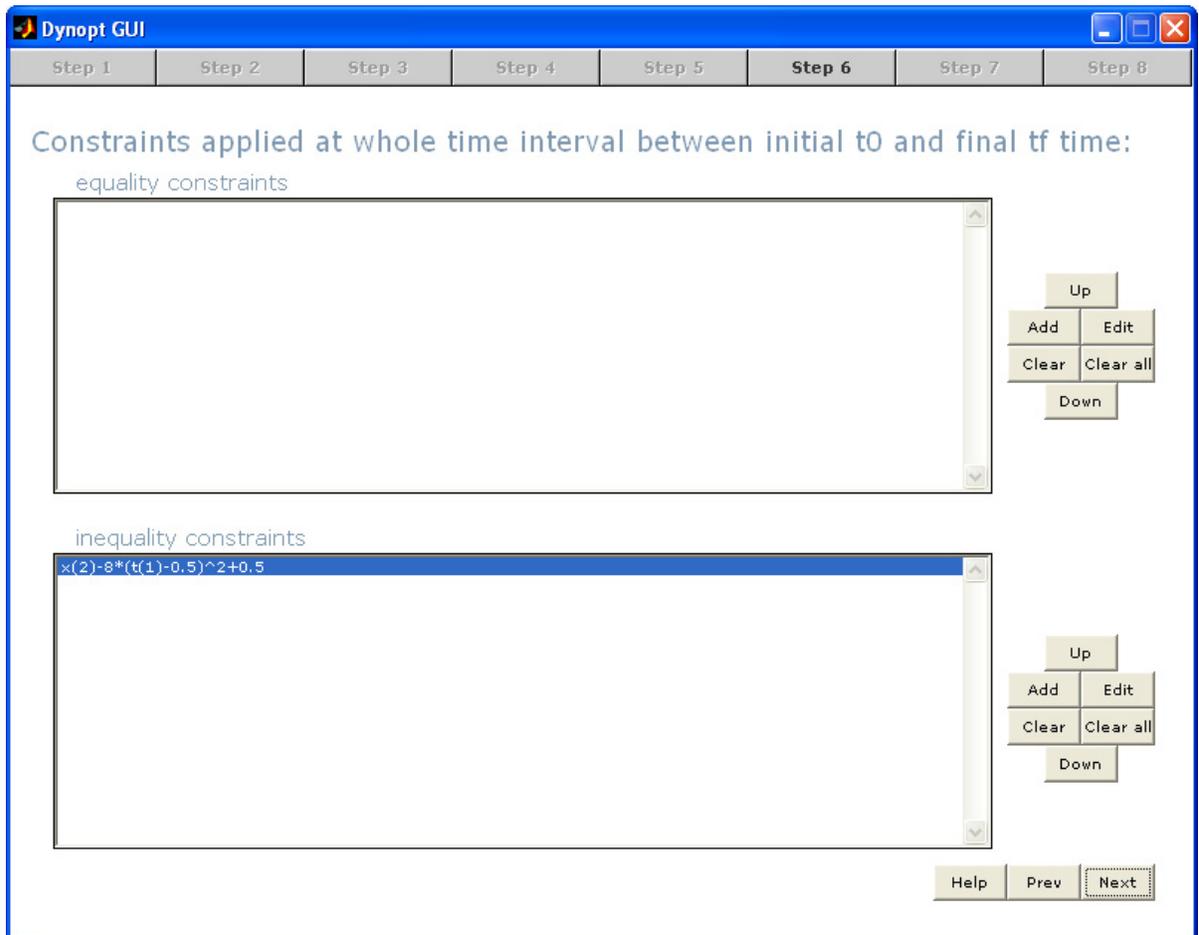


Figure A.8: Form for equality and inequality constraints between initial and final time

The image shows a screenshot of the Dynopt GUI. The window title is "Dynopt GUI". At the top, there is a progress bar with eight steps: Step 1, Step 2, Step 3, Step 4, Step 5, Step 6, Step 7 (which is currently selected and highlighted), and Step 8. Below the progress bar, the main content area is titled "Constraints applied at final time t_f ".

There are two main sections for inputting constraints:

- equality constraints:** A large empty text area with a vertical scrollbar on the right. To its right is a control panel with buttons: "Up", "Add", "Edit", "Clear", "Clear all", and "Down".
- inequality constraints:** A second large empty text area with a vertical scrollbar on the right. To its right is an identical control panel with buttons: "Up", "Add", "Edit", "Clear", "Clear all", and "Down".

At the bottom right of the main content area, there are three buttons: "Help", "Prev", and "Next".

Figure A.9: Form for equality and inequality constraints at final time

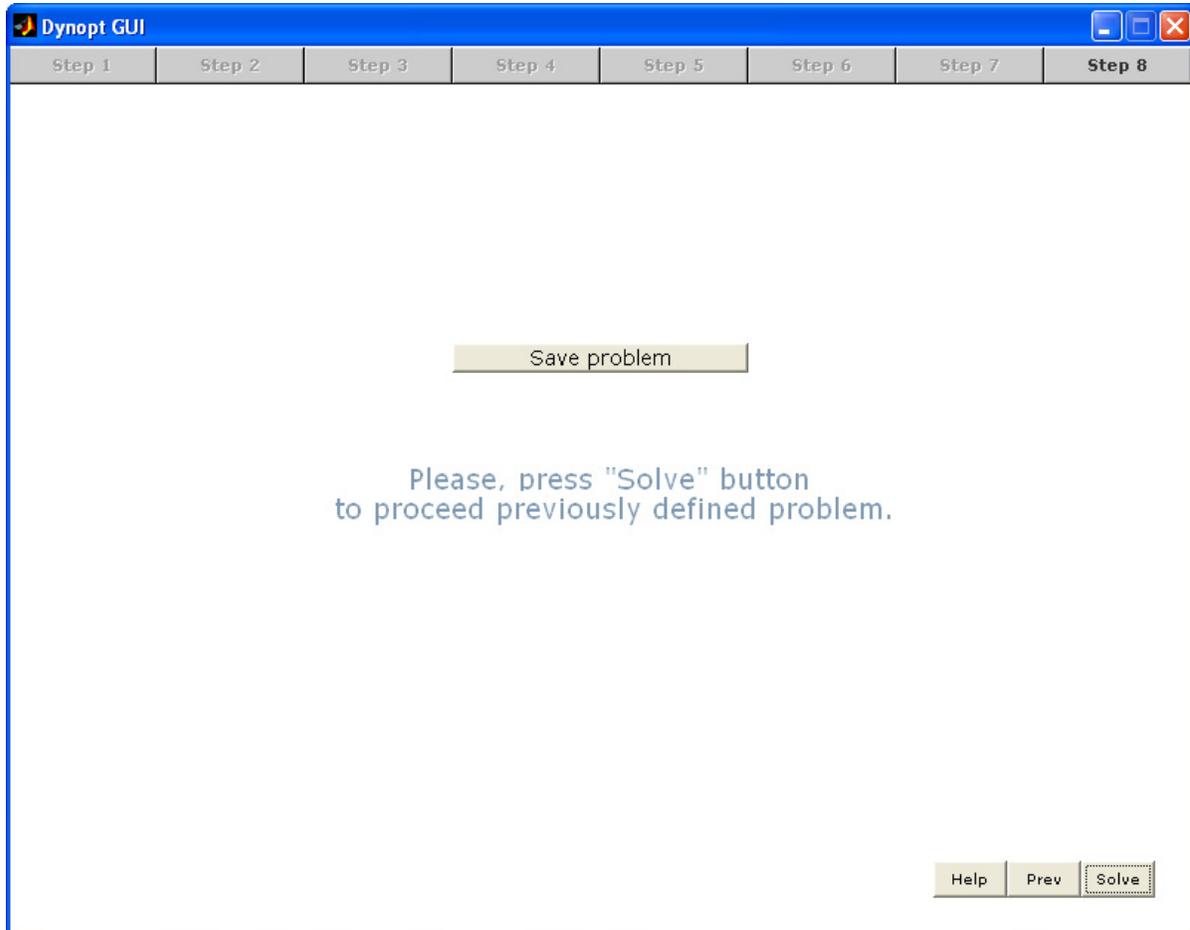


Figure A.10: Form to save and solve problem

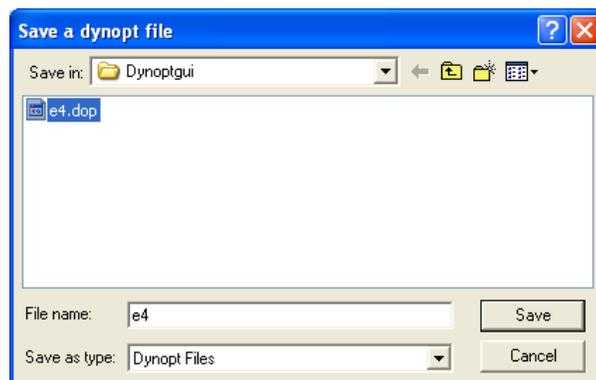


Figure A.11: Save dynamic optimization problem dialog