COMPRESSIVE CONTROL-VECTOR PARAMETERIZATION WITH DISCRETE COSINE TRANSFORM FOR THE SOLUTION OF OPTIMAL-CONTROL PROBLEMS

by

Ahmet Hallaçeli

B.S., Chemical Engineering, Middle East Technical University, 2015

Submitted to the Institute for Graduate Studies in

Science and Engineering in partial fulfillment of

the requirements for the degree of

Master of Science

Graduate Program in Chemical Engineering

Boğaziçi University

2017

ACKNOWLEDGEMENTS

Firstly, I wish to express my gratitude to my advisor Prof. Dr. Uğur Akman, not only for allowing to work with him but also for his infinite patience, support and motivation through this thesis work. It has been a great and wonderful experience working with him.

I would like to thank my thesis committee members: Assoc. Prof. Dr. Burak Alakent and Assoc. Prof. Dr. Devrim Barış Kaymak for kindly accepting to be in my committee and for their insightful comments.

I would like to mention my appreciations to Assoc. Prof. Dr. Burak Alakent for his permission to use his computers in the Process Systems Engineering Laboratory at the Department of Chemical Engineering.

My deep appreciations to all professors at the Middle East Technical University for the education standards they provided and for their support throughout my undergraduate years.

My sincere thanks goes to my dear friends at Boğaziçi University for their support and solidarity.

Last but not the least, I would like to thank my family: my parents and my sister for supporting me spiritually throughout the writing of this thesis and my life in general.

ABSTRACT

COMPRESSIVE CONTROL-VECTOR PARAMETERIZATION WITH DISCRETE COSINE TRANSFORM FOR THE SOLUTION OF OPTIMAL-CONTROL PROBLEMS

In this thesis, a novel numerical method for the solution of open-loop optimal-control problems is proposed. The method combines the flexibility of the standard Control-Vector Parameterization (CVP) technique with the compressive power of the Discrete Cosine Transform (DCT). Thus, the proposed method is termed as the Compressive Control-Vector Parameterization with Discrete Cosine Transform (CVP-DCT). In the CVP-DCT method, the control input is parameterized in terms of only few DCT Coefficients (DCTCs). The CVP-DCT method transcribes the optimal-control problem to a Nonlinear Programming (NLP) problem where the coefficients selected from the early elements of the DCTC vector are the optimization decision variables. Terminal and path constraints, as well as control bounds, are handled by the penalty-function method. Several problems are solved using the CVP-DCT, standard CVP, and Control-Vector Optimization (CVO) methods for comparison and demonstration of the pros and cons of the proposed CVP-DCT method. The method does not require a priori knowledge of the shape and complexity of the control trajectory and it can be used in any optimal control problem without prespecification. With only a few parameters, the CVP-DCT method can provide a good initial-guess trajectory to other sophisticated optimal control software packages. Especially if the control trajectory is smooth, the CVP-DCT method can provide solutions which are very close to the global solution using just a few decision variables. The performance, required number of DCTCs, and number of optimization decision variables of the method is independent of the dimension of the states and the number of time grids. Even very few DCTCs are enough for the reconstruction of the control vector to hundreds or even thousands of time grids without affecting the CPU time noticeably. Therefore, the proposed method is a viable technique for the fast solution of generic open-loop optimal-control problems with efficient low-dimensional parameterization.

ÖZET

OPTİMAL DENETİM PROBLEMLERİNİN AYRIK KOSİNÜS DÖNÜŞÜMÜYLE KOMPRESİF DENETİM VEKTÖRÜ PARAMETRELENDİRMESİ YÖNTEMİYLE ÇÖZÜMÜ

Bu tezde, açık döngülü optimal denetim problemlerinin çözümü için özgün bir sayısal yöntem sunulmuştur. Bu yöntem, standart "Denetim Vektörü Parametrelendirmesi" (DVP) yönteminin esnek kullanım özelliğiyle Ayrık Kosinüs Dönüşümü'nün (AKD) güçlü sıkıştırma özelliğini birleştirmiştir. Bu sebeple "Ayrık Kosinüs Dönüşümüyle Kompresif Denetim Vektörü Parametrelendirmesi" (DVP-AKD) olarak adlandırılmıştır. DVP-AKD'de parametrelendirme, yalnızca birkaç Ayrık Kosinüs Dönüşü Katsayısı (AKDK) kullanarak, sadece denetim değişkenlerine uygulanır. DVP-AKD yöntemi, optimal denetim problemlerini Doğrusal Olmayan Programlama (DOP) problemlerine dönüştürmektedir ve AKDK vektörünün ilk birkaç elemanını karar değişkeni olarak kullanmaktadır. Durum ve denetim değişkenlerinin kısıtlamaları ceza fonksiyonu yöntemiyle ele alınmaktadır. DVP-AKD'nin performansını ölçmek için, çeşitli problemler standart DVP ve "Denetim Vektör Optimizasyonu" (DVO) yöntemleriyle de çözülmüştür. Üç farklı yöntemin kıyaslaması yapılarak DVP-AKD yönteminin iyi ve kötü yanları belirtilmiştir. DVP-AKD yöntemi, denetim değişkenlerinin şeklinin veya karmaşıklığının önbilgisine gerek duymamaktadır. Bundan ötürü, bu yöntem herhangi bir optimal denetim problemine uygulanabilir. Sadece birkaç parametreyle DVP-AKD yöntemi, diğer yöntemler için tutarlı bir ön tahmin sağlamaktadır. Denetim dalgalı bir şekle sahip değilse, DVP-AKD sadece birkaç katsayı ile analitik çözüme çok yakın sonuçlar elde edebilmektedir. DVP-AKD'nin performansı, gerekli karar ve durum değişkenleri sayılarından ve ayrıklaştırılmış zaman elemanı sayısından bağımsızdır. Çok az AKDK dahi karar değişkeni davranışının yüzlerce hatta binlerce ayrık zaman elemanı ile oluşturulması için yeterli olup işlemci zamanı ve ayrık zaman eleman sayısına bağlı değildir. Bundan dolayı, sunulan bu yöntem, verimli parametre sıkılaştırması sayesinde, açık döngülü optimal denetim problemlerinin hızlı çözümlerini elde edilebilmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

c	Discrete cosine transform coefficients
C	Inequality path constraints
c(k)	Discrete cosine transform coefficients
d	Set of decision variables
E	Equality constraints
H	Hamiltonian
Hp	Hurtz parameter
J	Performance ındex
J*	Optimal performance ındex
k	Matrix index
L	Time dependent cost function
m	Number of control variables
M	Discrete cosine transform matrix
n	Number of state variables
N	Number of elements
p	Parameter
t	Time
t_0	Initial-time
t_f	Final-time
u	Control input variable
u	Control vector
u(t)	Time-variant control action
u(t,x)	Time-and-state-variant control action
u*	Optimal control input
w(k)	Weight parameter
x	State variable
x(n)	Data matrix
x*	Optimal state variable
δ	Variation function

Δ	Difference function
ε	Step size
λ	Costate variable
μ_e	Equality constraint penalty parameter
μ_{pc}	Path constraint penalty parameter
μ_s	Smoothing penalty parameter
μ_{ul}	Upper-lower bound penalty parameter
Φ	Final-time cost function
$\boldsymbol{\Psi}$	Terminal (final-time) inequality conditions
$\boldsymbol{\Omega}$	Terminal (final-time) equality constraints

LIST OF ACRONYMS/ABBREVATIONS

BCI	Boundary Condition Iteration
BDF	Backward Differentiation Formula
CV	Calculus of Variations
CVI	Control Vector Iteration
CVO	Control Vector Optimization
CVP	Control Vector Parameterization
CVP-DCT	Compressive-Control Vector Parameterization
DCT	Discrete Cosine Transform
DCTC	Discrete Cosine Transform Coefficients
DE	Differential Evolution
ECP	Equality Constraint Penalty Function
FBM	Fractional Brownian Motion
i-DCT	Inverse Discrete Cosine Transform
IVP	Initial Value Problem
Lfbm	Length of Fractional Brownian Motion Signal
Ndctc	Number of Discrete Cosine Transform Coefficients
NLP	Nonlinear programmnig
Ntgrid	Number of time grids
OCT	Optimal Control Theory
ODE	Ordinary Differential Equations
PCP	Path Constraint Penalty Function
PMP	Pontryagin's Maximum Principle
RT	Reconstructed Trajectory
SP	Smoothing Penalty Function

SQP	Sequential Quadratic Programming
TPBVP	Two Point Boundary Value Problems
u_{grid}	Control variables on discretized time grids
ULP	Upper-Lower Bound Penalty Function
umax	Maximum value ot the control variable
umin	Minimum value of the control variable
LB	Lower Bound
UB	Upper Bound

1. INTRODUCTION

Most real world problems have several solutions, and some may have infinite number of solutions. Optimization is a mathematical tool which aims to find the best solution by finding the extremum point (maximum or minimum) of a performance criterion. The focus of this thesis work is dynamic optimization towards chemical engineering applications. In chemical engineering applications, determination of the optimal operating policies are the key elements to provide safe, profitable and environmentally-friendly processes. Furthermore, optimization has become a dominant technology for chemical industries to remain competitive in the market (Biegler, 2010).

Optimization is surveyed under two headlines which are “static optimization” and “dynamic optimization”. In static (steady-state) optimization, the system models are time invariant and formed by algebraic equations only. However, in dynamic optimization, objective function is a function of time (hence it is called objective functional) and the system models are represented by time-dependent models in the form of ODEs or PDEs. Dynamic optimization is also called "optimal control". An optimal control input is a control function of time, u(t), that leads a dynamic system to optimize (maximize or minimize) a performance criterion through satisfying the physical constraints (Kirk, 2004). Choosing a time-variant control action, u(t), can be accomplished in two different ways. If the control vector is a function of time only, u(t), it is called open loop control, if the control vector is a function of state variables as well, u(t,x), it is called closed-loop or feedback control (Upreti, 2012). In this thesis work, only the open-loop optimal-control problems were investigated where the control vector is a function of time only, u(t).

Optimal-control problems can be solved by either analytical or numerical methods. Numerical solution can be applicable for any problem whereas the availability of analytical solution depends on the complexity of the mathematical model of the problem. There are large variety of computational techniques available for determining the control policies.

Solutions of dynamic optimization problems appeared in the 17th century under favor of the advancements in the field of "calculus of variations". The necessary condition of optimality can be derived through the rules of calculus of variations which provide the analytical solution. If the complexity and dimension of the model is low (linear model or low number of dependent variables, i.e., state variables) and the derivation of the necessary condition of optimality is undemanding, the analytical solution is available, and it provides the guaranteed global optimum solution. For more complex models (nonlinear model or high number of dependent variables, i.e., state variables), analytical methods require immense amount of symbolic algebra that may easily get intractable, and thus, usually analytical solutions are not possible in practice (Srinivasan *et al*., 2003) Therefore, numerical methods are widely used because their implementation are more amenable for practical applications.

Numerical methods are either based on "indirect" or "direct" approaches. Direct methods are also referred to as "discretize-then-optimize" methods, because, either only the control variables or both the state and control variables are first discretized to create a finite-dimensional problem, and then, are solved by an optimization algorithm. On the other hand, in indirect methods, continuous optimality conditions are set then followed by a discretization strategy. Therefore, indirect methods are also referred to as "optimize-then-discretize" methods (Betts, 2010). Indirect methods are relatively older methods; they were developed in the 1950s (cold-war period). These methods utilize optimality conditions either derived through the calculus of variations or Pontryagin's Minimum Principle (Bryson and Ho, 1975; Spangelo, 1994). Direct methods have become more and more available and popular due to advancements in computer technology which increased memory capacity and computational power tremendously. In direct numerical methods, infinite-dimensional optimal-control problems are transformed to finite dimensional nonlinear programming (NLP) problems. Direct methods discretize both the state and control variables (simultaneous method) or only the control variables (sequential method). The simultaneous method (also known as complete parameterization approach) has emerged in the late 1960s by (Polak, 1969; Canon *et al*, 1969). At the early times, the parameterization of state ODEs was implemented by finite-difference techniques (Sargent, 2000). The simultaneous methods were improved by the employment of the collocation methods, first by (Tsang *et al*, 1975) and followed by many other researchers (Cuthrell and Biegler, 1987). Currently,

there are variety of simultaneous methods based on different collocation strategies such as orthogonal collocation (Oh and Luus, 1977), collocation on finite elements (Cizniar *et al*., 2005) or pseudospectral collocation (Rutquist *et al*., 2010). A review of simultaneous approach was provided by (Biegler, 2007).

The sequential method has become popular in chemical engineering problems thanks to developments by Sargent and co-workers (Gritsis, 1990; Sargent and Sullivan, 1978; Morison and Sargent, 1986; Vassiliadis, 1993). In sequential methods, control variable is approximated over time domain by finite set of local functions or global functions. The Control Vector Parameterization (CVP) method uses global functions, that operate from initial time to final time, for approximation and it may work very well when the correct functional form is utilized. However, its performance strictly depends on the shape (complexity) of the control trajectory. Therefore, the local functions, were replaced by the local polynomials that operate over the individual sub-sections (zones) only, in the interval from initial time to final time. In Control Vector Optimization (CVO), piecewise-constant parameterization approach is used. The time domain is divided into usually equal-length time intervals and the state values (obtained from the solution of model ODEs) on these time grids are used as optimization decision variables. The CVO can be applied to any optimal-control problem. However, even if time discretization is handled judiciously it suffers from curse of dimensionality.

The main purpose of this thesis work is to utilize the advantages of the sequential method and at the same time eliminate its disadvantages, especially regarding curse of dimensionality. The proposed method, Compressed Control-Vector Parameterization with Discrete Cosine Transform (CVP-DCT) can be considered as a global parameterization technique as CVP, yet it can be implemented to any problem as CVO. The Discrete Cosine Transform (DCT) is a data-compression tool invented by Ahmed et. al (1974) for image- / signal-processing purposes. However, in the CVP-DCT method, the parameter compression attribute of DCT is used to compress the parameterization of the unknown / unspecified control vector. The values of the control-vector over the discretized time domain are compressed into the DCT coefficients which are time-independent parameters. The

decompression (unfolding) of the parameters by inverse-DCT (i-DCT) creates (restores / reconstructs) the control vectors on these time grids. Therefore, the main purpose is to parameterize the control trajectory, u(t), with much fewer number of decision variables and without a priori knowledge on the shape and complexity of the control trajectory.

In Chapter 2 of this thesis, the fundamentals and general properties of the Discrete Cosine Transform are given. Furthermore, in Section 2.1, the compression property of DCT is exposed and the compressive parameterization of representative control trajectories, which is the main idea of this thesis work, are examined.

In Chapter 3, the optimal-control theory and general classification of optimal-control problems are briefly reviewed.

In Chapter 4, two mainstream sequential direct methods to solve optimal-control problems are presented. In Section 4.1, the control-vector parameterization method, and in Section 4.2, the control-vector optimization method is discussed. The algorithms, and advantages and disadvantages of these methods are presented. The penalty-function method, which is chosen to handle constrained optimal-control problems, is introduced in Section 4.3.

In Chapter 5, the focus is on the novel method of this thesis work, named as the Compressed Control-Vector Parameterization with Discrete Cosine Transform (CVP-DCT). The CVP-DCT algorithm and its characteristic features are provided in this section.

In Chapter 6, several optimal-control problems with different aspects are solved with the CVP, CVO, and CVP-DCT methods. In Section 6.1, an optimal-control problem with final-time equality constraints (Problem 1) is solved. Only for Problem 1, the analytical solution and an indirect solution are provided to compare with the numerical methods of CVP, CVO, and CVP-DCT. Problems in Sections 6.2, 6.3, and 6.4 have only upper and

lower control-bound constraints. In Section 6.5 and 6.6, problems with state path constraints are solved. Finally, in Section 6.7. a problem with two control inputs with upper and lower control-bound constraints is provided.

In Chapter 7, the overall summary of this thesis work, together with the pros and cons of the suggested CVP-DCT method, is provided and the recommendations for future studies are given.

All computations were done with MATLAB. Sample MATLAB codes for each of the algorithms used in this thesis work are provided in the Appendix section.

2. THE DISCRETE COSINE TRANSFORM

Discrete Cosine Transform (DCT) was invented by Ahmed *et al*. (1974). It is a widely-used tool in signal/image processing. Due to its compressive approximation power, data-independent basis, and low computational complexity, DCT is a widely used linear and invertible transform method (Strang, 1999). DCT is a Fourier-based method that uses cosine functions as basis. However, DCT is not the real part of discrete Fourier transform. There are eight standard variants of DCT labeled from DCT-I to DCT-VIII. In this thesis work, the most widely used one, the DCT-II, was used to transform one-dimensional data, which are the control, u(t), values at uniform time grids. The DCT is formulated as follows (Ahmed *et al*., 1974).

$$c(\mathrm{k}) = \mathrm{w}(\mathrm{k}) \sum_{\mathrm{n}=1}^{\mathrm{N}} \mathrm{x}(\mathrm{n}) \cos\left[\frac{\pi(2\mathrm{n}-1)(\mathrm{k}-1)}{2\mathrm{N}}\right], \qquad \mathrm{k} = 1, 2, \ldots, \mathrm{N} \tag{2.1}$$

$$\mathrm{w}(\mathrm{k}) = \begin{cases} \sqrt{1/\mathrm{N}}, & \mathrm{k} = 1 \\ \sqrt{2/\mathrm{N}}, & 2 \leq \mathrm{k} \leq \mathrm{N} \end{cases}$$

Both input and output of the transformation are real numbers. The input, $\mathrm{x}(\mathrm{n})$, is transformed into a set of coefficients so called Discrete Cosine Transform Coefficients (DCTC) which are denoted as $\mathrm{c}(\mathrm{k})$. Moreover, DCT is an invertible transform. The inverse transformation of DCT (i-DCT) is given as follows.

$$\mathrm{x}(\mathrm{n}) = \mathrm{w}(\mathrm{k}) \sum_{\mathrm{k}=1}^{\mathrm{N}} \mathrm{c}(\mathrm{k}) \cos\left[\frac{\pi(2\mathrm{n}-1)(\mathrm{k}-1)}{2\mathrm{N}}\right], \qquad \mathrm{n} = 1,2, \ldots, \mathrm{N} \tag{2.2}$$

$$\mathrm{w}(\mathrm{k}) = \begin{cases} \sqrt{1/\mathrm{N}}, & \mathrm{k} = 1 \\ \sqrt{2/\mathrm{N}}, & 2 \leq \mathrm{k} \leq \mathrm{N} \end{cases}$$

The DCT and i-DCT transforms were implemented by the “dct” and “idct” functions of MATLAB.

Furthermore, DCT is a linear transform and its matrix form is given as follows (Jain, 1989).

$$\boldsymbol{c} = \mathbf{M}\,\mathbf{x} \tag{2.3}$$

$$\text{where} \begin{cases} \mathbf{x}\text{: data matrix (N x 1)} \\ \mathbf{M}\text{: discrete cosine transform matrix (N x N)} \\ \mathbf{c}\text{: discrete cosine transform coefficients (N x 1)} \end{cases}$$

The DCT matrix, **M**, is a real and orthogonal matrix. Therefore,

$$\mathbf{M^{-1}} = \mathbf{M^{T}} \tag{2.4}$$

The original data matrix can be reconstructed from its DCTCs as;

$$\mathbf{x} = \mathbf{M^{-1}c} \tag{2.5}$$

One of the main features of DCT is that it provides very effective information compression. It has an excellent energy compaction, especially for highly correlated data (e.g., for low-frequency nonstationary time-series). The earlier (first few) elements of **c** have values larger in magnitude than the following elements and most of the later elements are zeros or much smaller in magnitude (Salomon, 2008). In other word, the magnitude of the DCT coefficients decay fast towards zero. For high-frequency stationary time series, however, the DCT coefficients do not decay fast enough, i.e., they persist to exist in large magnitudes throughout the vector **c**, and thus, DCT does not provide much energy compaction over the early elements only.

Equation (2.1) shows that 1st DCTC (k=1) has a cosine term that is equal to 1. Therefore, the value of 1st DCTC only depends on the arithmetic mean of $\mathbf{x}(k)$. Furthermore, a dataset of constant numbers can be represented with a single DCTC as demonstrated with the following example. Let the 10-element vector **x** has a constant trajectory given as $\mathbf{x}(k) = [10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ \ 10]$. The DCTC of vector **x**, becomes the 10-element DCTC vector, $\mathbf{c}(k) = [31.62\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$. That is, the information content of array **x** was compressed to only one number, the first DCTC which is 31.62. This behavior is independent of the cardinality (number of elements) of such a constant **x** vector. In other words, it is

sufficient to know/store only the first DCTC to recover (reconstruct) any such constant **x** vector of arbitrary cardinality.

In the following section, the compaction property of DCT was investigated using different data series. These series (discretized functions) can be considered to be typical control trajectories in optimal-control problems.

2.1. Compressive DCT Parameterization of Representative Control Trajectories

In this section, we investigate the compressive parameterization potential of DCT on several discretized functional data that may be considered as representatives for typical optimal control trajectories, u(t). Firstly, we selected normalized continuous trajectories, u, then these functions were mapped into normalized linear vectors as **u** with N elements. DCT of **u**'s were taken and the sequential DCTC data were represented by "stem" function of MATLAB. This representation way was selected in order to better portray the order of magnitude of the DCTCs. Different number of DCTCs (*Ndctc*) were selected as the reconstruction parameter. As given in Equation (2.3) and Equation (2.5), the i-DCT of the full DCTCs yields the original data identically (with zero reconstruction error). On the other hand, taking only the first few (*Ndctc*) coefficients and setting the rest of the DCTCs to zero, so called "zero-padding", is an approximate / compressive reconstruction method. Zero-padding before i-DCT brings error, and this error is defined as the "reconstruction error".

In this thesis, for a DCTC vector of N elements, the zero-padding were done by replacing the [N–*Ndctc*] elements after the first *Ndctc* elements with zero values. The i-DCT of thus created vectors with N elements were used to reconstruct the approximate trajectories of length N of the original trajectory. When *Ndctc* = N, this procedure yields a perfect decompression (perfect reconstruction), i.e., reconstructed trajectory is the same as the original trajectory as given in Equation (2.3) and Equation (2.5). Reconstruction with only one DCTC, i.e., [N−1] zero padded elements, can only create an array of constant numbers (constant horizontal trajectory, i.e., horizontal line). Hence, in the following examples, the lowest *Ndctc* used was taken as two. The reconstructed trajectories (RT) with different *Ndctc*

were compared. Furthermore, reconstruction error was defined as the Euclidian norm of the difference between the original and reconstructed data.

An example function in the interval [0,1] sampled with N=100 points, its full set of DCTCs, and its reconstructions with *Ndctc*=N are depicted in Figure 2.1. It should be noted that the *x*-axis of the subplot that shows the DCTCs was made logarithmic in order to provide a better resolution for the early DCTCs.

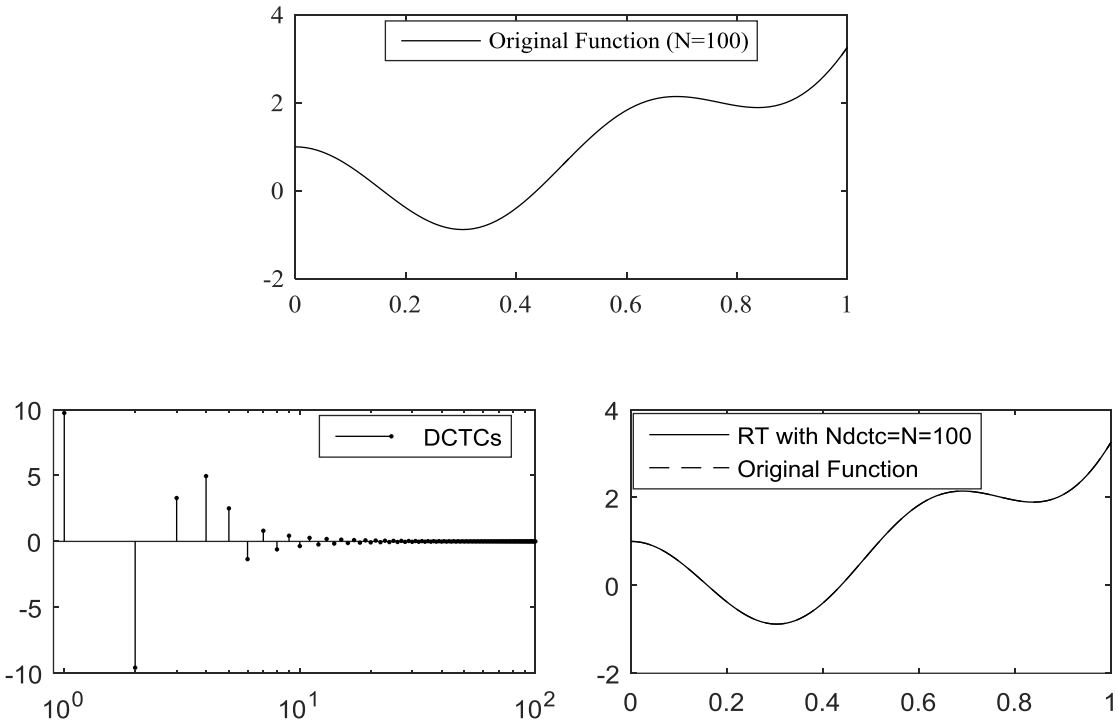


Figure 2.1. An example function sampled with N=100, its DCTCs, and its reconstruction with *Ndctc*=100.

Taking *Ndctc*=100 yielded a perfect reconstruction (error norm is exactly equal to zero) as previously discussed. Furthermore, reconstructions with different *Ndctc* (2-10) were also done to analyze compression power of DCT. More DCTC leads to more accurate reconstruction. The error norms of the reconstruction with corresponding *Ndctc* values are given in Table 2.1. The DCTCs used and reconstructed trajectories with *Ndctc*=10, 8, 6, and 4 are given in Figures 2.2 to 2.5. For Figures 2.2 - 2.5, it should be noted that among the DCTCs of the original function given in Figure 2.2, the DCTCs on the right-hand side of the

vertical dashed lines have zero values due to zero-padding before the reconstruction via i-DCT

Table 2.1. Error norms in reconstruction of the example function.

Ndctc	2	3	4	5	6	7	8	9	10
Error	6.71	5.85	3.11	1.85	1.27	0.98	0.77	0.64	0.53

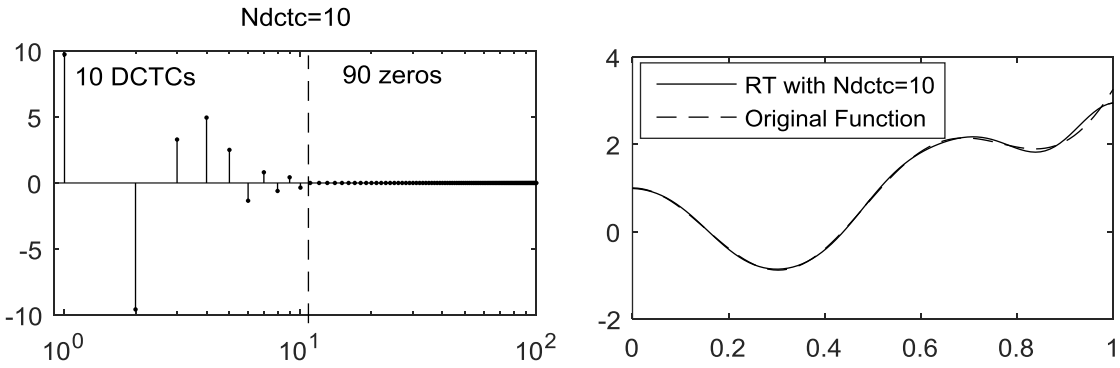


Figure 2.2. Reconstruction of the example function with *Ndctc*=10.

Since the 11th and subsequent DCTCs had magnitudes already very close zero (see Figure 2.1), the reconstructed trajectory is almost identical to the original trajectory.

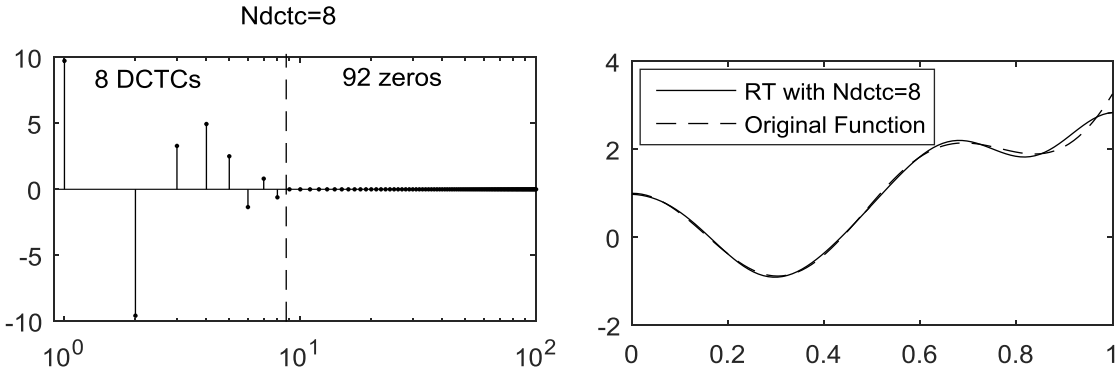


Figure 2.3. Reconstruction of the example function with *Ndctc*=8.

Since the DCTCs after 8th location are small but not extremely close to zero, the difference between the original and the reconstructed trajectories with *Ndctc*=8 have a slight

difference. In other words, the zeroing of the 9th, and 10th DCTCs is responsible for the difference between the reconstructions with *Ndctc*=8 (Figure 2.3) and *Ndctc*=10 (Figure 2.2).

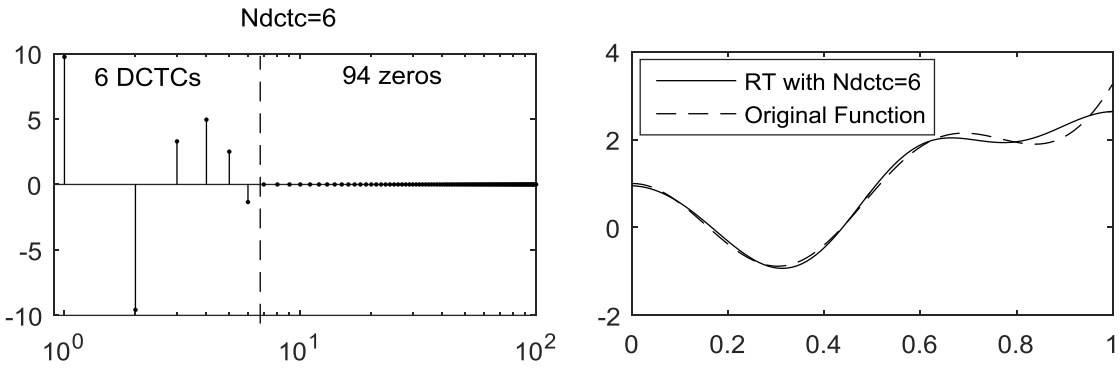


Figure 2.4. Reconstruction of the example function with *Ndctc*=6.

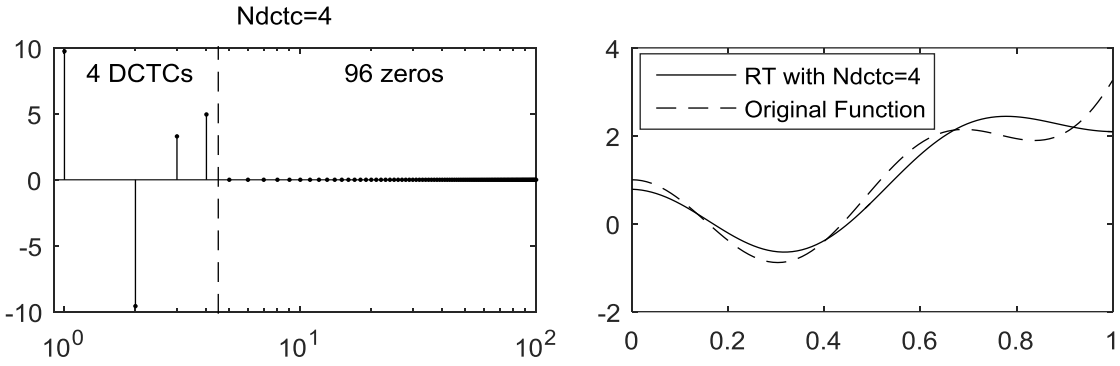


Figure 2.5. Reconstruction of the example function with *Ndctc*=4.

Reconstruction with *Ndctc*=4 (Figure 2.5) yielded a rough approximation relatively to the trajectory obtained with *Ndctc*=6 (Figure 2.4) because the 5th and 6th DCTCs were not close to zero (see Figure 2.1).

2.1.1. Linear Function

Let $u(t) = a + bt$, be a normalized linear function sampled with 10 elements, N=10. The linear function was reconstructed with different *Ndctc* values. The original and reconstructed trajectories with *Ndctc* = 2, 3, 4, and 5 are given in Figure 2.6. It should be

noted that a linear trajectory cannot accurately be reconstructed with 2 or 3 DCTCs. The Euclidean norm of the reconstruction errors are given in Table 2.2.

Table 2.2. Error norms in reconstruction of the linear function.

Ndctc	2	3	4	5	6	7	8	9	10
Error	0.114	0.114	0.038	0.046	0.038	0.015	0.015	0.004	0

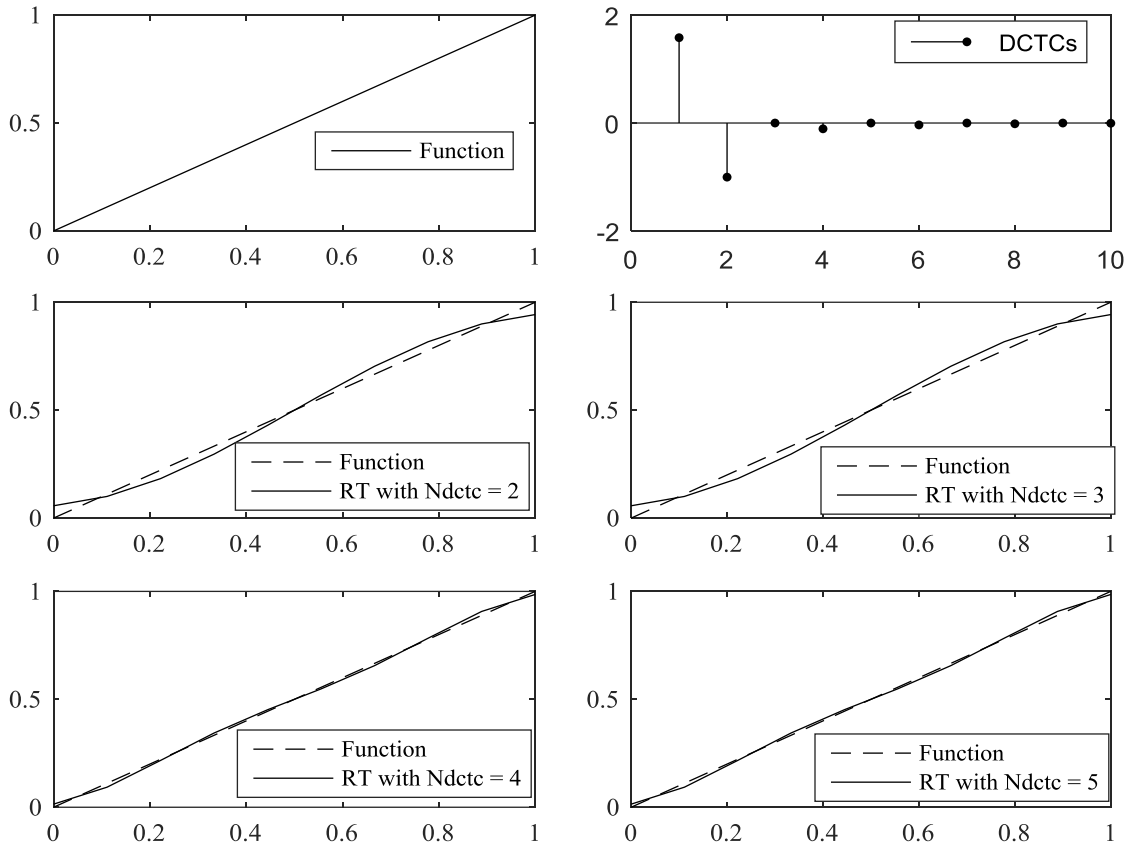


Figure 2.6. The linear trajectory sampled with N=10, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

2.1.2. Second-Order Polynomial

Let $u(t) = a + bt + ct^2$ be a second order polynomial function sampled with N=10. The reconstruction error norms are given in Table 2.3. The original and reconstructed trajectories with *Ndctc* = 2, 3, 4, and 5 are shown in Figure 2.7.

Table 2.3. Error norms in reconstruction of the second-order polynomial function.

Ndctc	2	3	4	5	6	7	8	9	10
Error	0.306	0.134	0.080	0.047	0.030	0.018	0.010	0.004	0

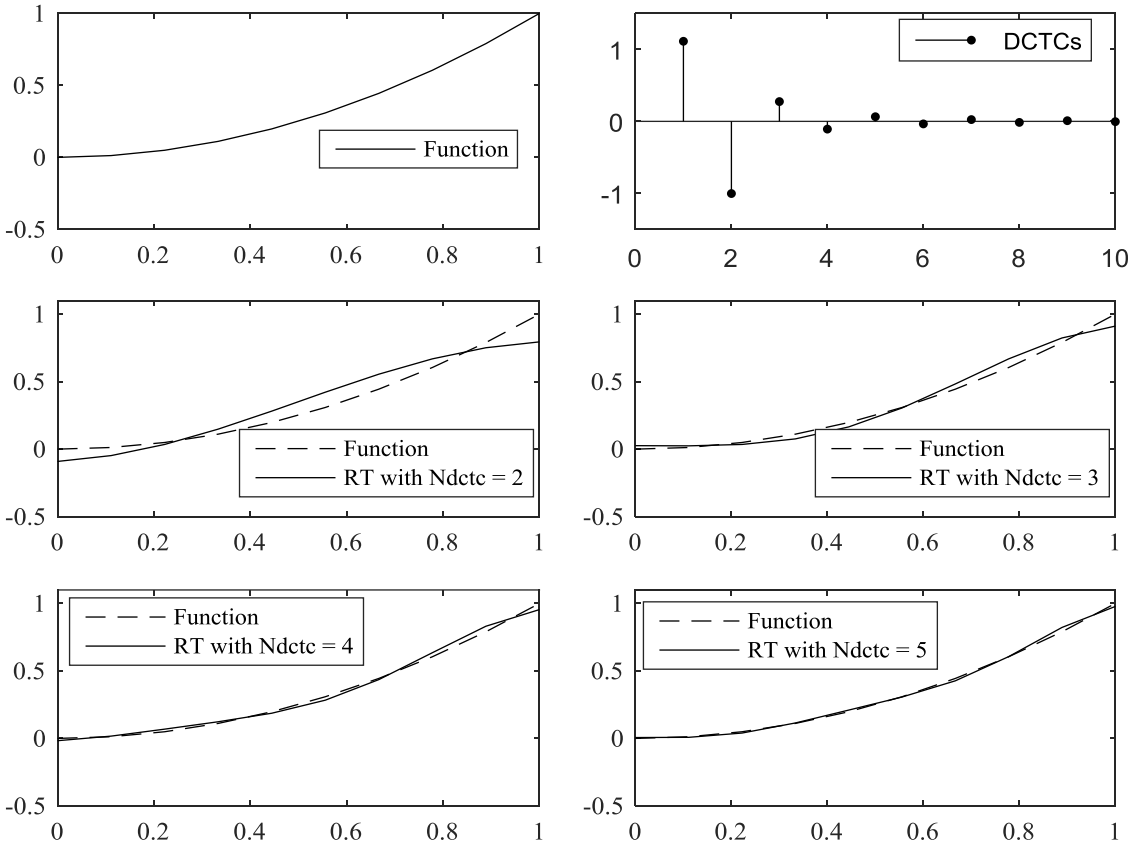


Figure 2.7. The 2nd order polynomial trajectory sampled with N=10, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

2.1.3. Cosine Function

Let $u(t) = \cos(10\,t)$ and N=10. The original and reconstructed trajectories with *Ndctc* = 2, 3, 4, and 5 are given in Figure 2.8. Furthermore, to analyze the effect of *Ndctc*/N ratio, the calculations were repeated by taking N = 100 samples from the function. The original and reconstructed trajectories with N=100 and with *Ndctc* = 2, 3, 4, and 5 are given in Figure 2.9.

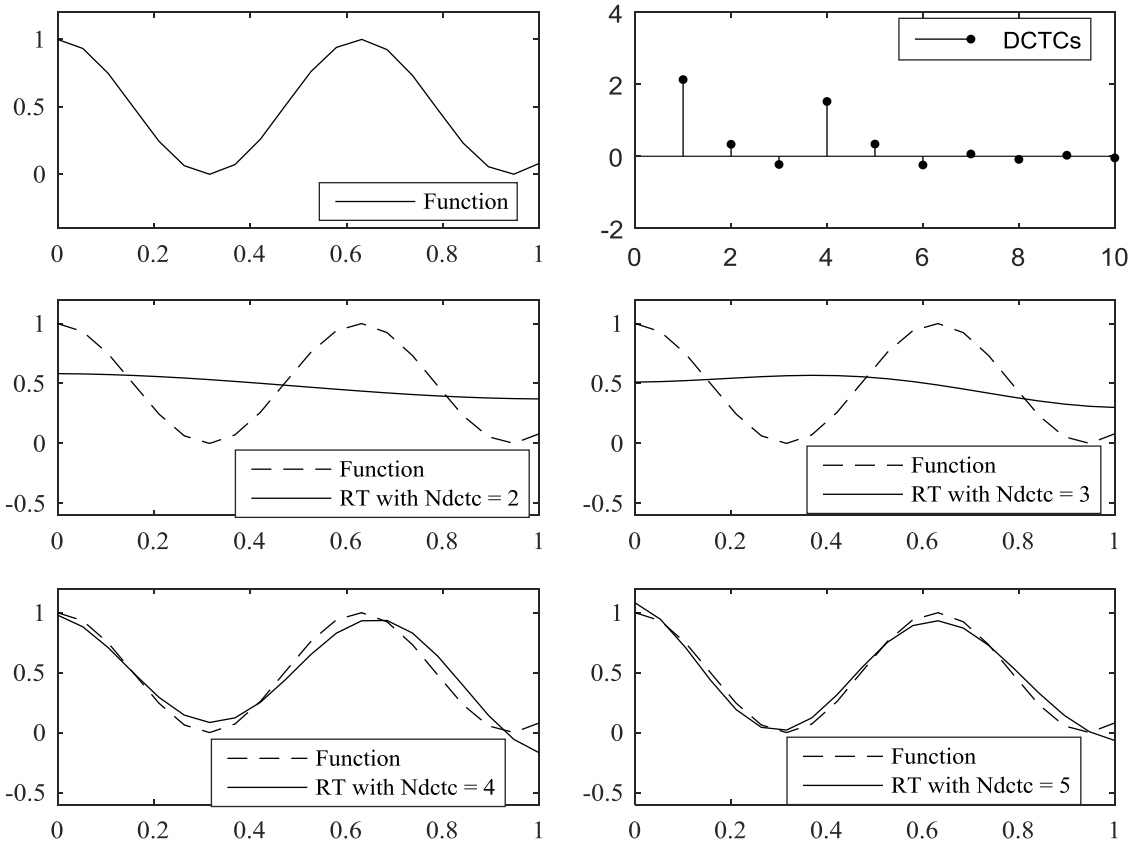


Figure 2.8. The cosine function trajectory sampled with N=10, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

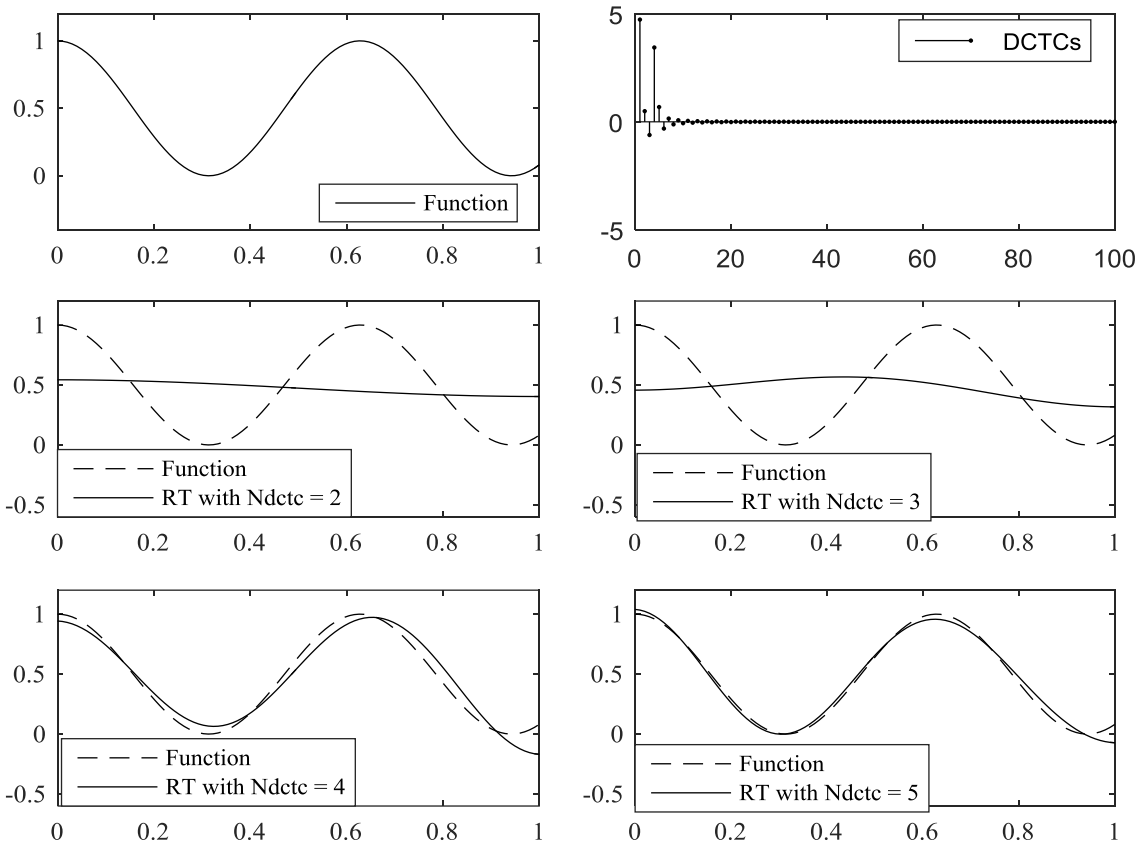


Figure 2.9. The cosine function trajectory sampled with N=100, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

The reconstruction error norms with both N=10 and N=100 samples are given in Table 2.4.

Table 2.4. Error norms in reconstruction of the cosine function.

Ndctc	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Error N=10	1.131	1.125	0.392	0.285	0.086	0.078	0.022	0.019	0
Error N=100	3.583	3.531	0.788	0.390	0.232	0.177	0.1307	0.109	0.880
Ndctc	20	30	40	50	60	70	80	90	100
Error N=100	0.088	0.028	0.014	0.08	0.004	0.003	0.002	0.001	0

According to the Table 2.4 and the Figure 2.9, reconstruction with *Ndctc*=2 and 3 did not yield a good approximation for this function. As the correlation between the elements of

sample decreases, reconstruction requires more *Ndctc*. Furthermore, at the same *Ndctc*, N=10 reconstructed more accurately than N=100. On the other hand, comparison of the same *Ndctc*/N ratio, (e.g. comparing *Ndctc* = 5, N=10 and *Ndctc*=50, N=100) showed that higher N yielded better reconstruction. When *Ndctc* $\geq$ 20 for N=100 case, complete reconstruction was almost achieved.

2.1.4. Piecewise Linear Function I

Let u(t) be a piecewise linear function that consists of two linear segments. The vector of N elements is given as $\mathbf{u} = [1 \;\; 0.75 \;\; 0.5 \;\; 0.25 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0]$. The 5th and 6th DCTCs were equal to zero. Therefore, reconstruction was done with *Ndctc*=2, 3, 4, and 7. The reconstruction error norms are given in Table 2.5.

Table 2.5. Error norms in reconstruction of the piecewise linear function I.

Ndctc	2	3	4	5	6	7	8	9	10
Error	0.59	0.197	0.073	0.073	0.073	0.053	0.024	0.024	0

In this example, the difference between RT's with *Ndctc*=4 and *Ndctc*=7 are quite small and the trajectories are almost identical. The original and reconstructed trajectories are given in Figure 2.10.

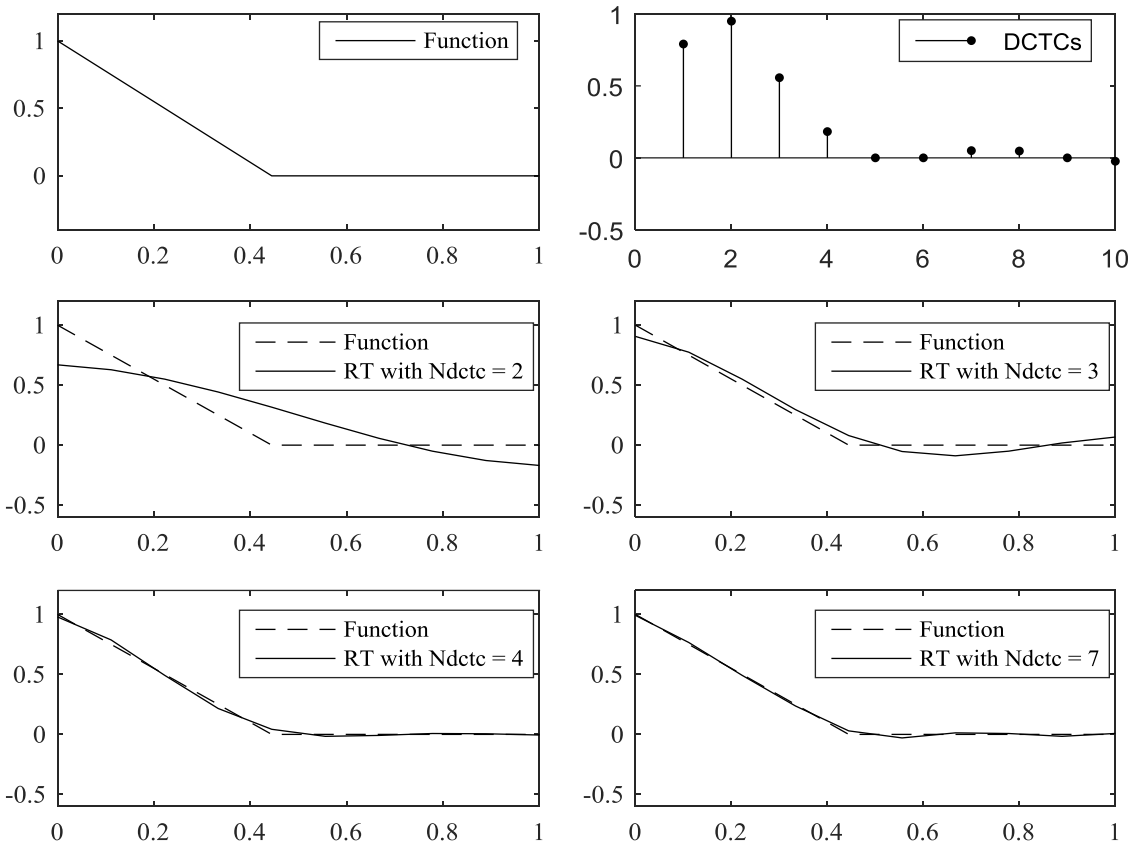


Figure 2.10. The piecewise linear function I trajectory sampled with N=10, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

2.1.5. Piecewise Linear Function II

Let u(t) be a piecewise linear function that consists of three linear segments. The vector of N elements is given as: $\mathbf{u} =$ [1 0.5 0 0 0 0 0 0 0.5 1]. The 2nd, 4th, 6th, 8th and 10th DCTCs were equal to zero. Therefore, reconstruction was done with *Ndctc*=3, 5, 7, and 9. The reconstruction error norms are given in Table 2.6. The original and reconstructed trajectories are given in Figure 2.11.

Table 2.6. Error norms in reconstruction of the piecewise linear function II.

Ndctc	2	3	4	5	6	7	8	9	10
Error	1.265	0.600	0.600	0.132	0.132	0.085	0.085	0	0

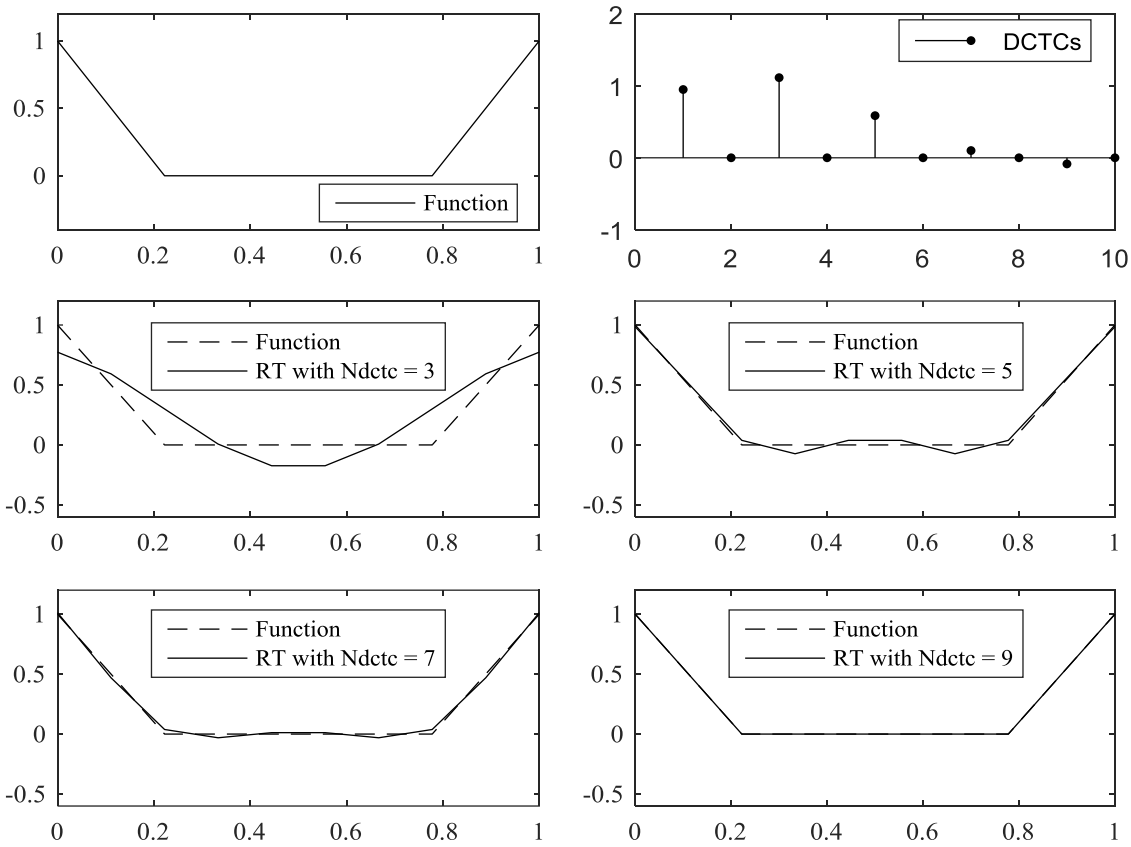


Figure 2.11. The piecewise linear function II trajectory sampled with N=10, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

2.1.6. Stepwise Function

Let u(t) be a vector of step inputs with N = 100 samples. Each step input has equal length but different magnitude. Reconstruction was done with *Ndctc* = 5, 10, 15, and 20. The reconstruction error norms are given in Table 2.7. The original and reconstructed trajectories are given in Figure 2.12.

Table 2.7. Error norms in reconstruction of the stepwise functions.

Ndctc	5	10	15	20	40	50	80	90	100
Error	1.904	1.291	1.029	0.873	0.509	0.506	0.288	0.201	0

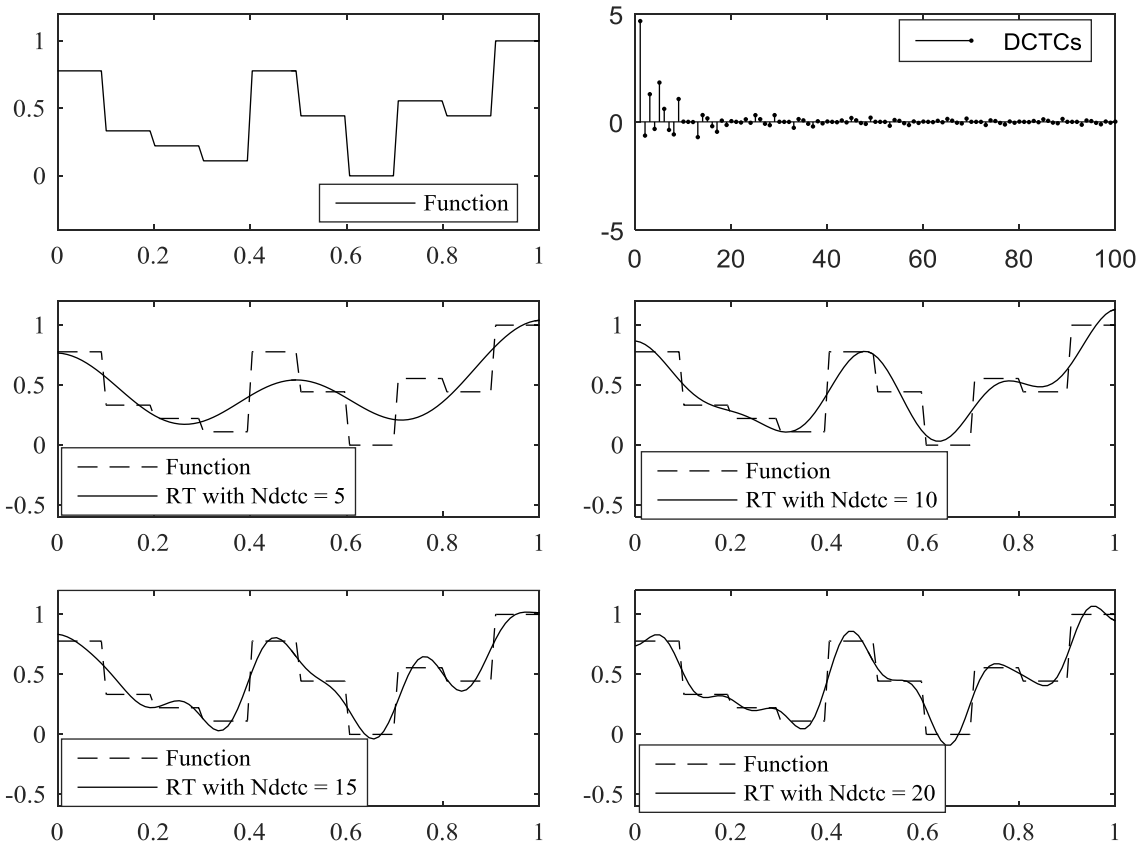


Figure 2.12. The stepwise function trajectory sampled with N=100, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

The reconstruction with *Ndctc*=20 provided a reasonably accurate decompression as given in Figure 2.12. It should be noted that 80 of the 100 DCTCs were taken as zero by zero-padding.

2.1.7. Fractional Brownian Motion Signal

In this section, much larger data sets obtained from the time series generated from the Fractional Brownian Motion (FBM) simulations were considered with N=100, 1000, 10000, and 100000 data points. The "wfbm" function of MATLAB was used to generate the data sets. The Hurtz parameter of the FBM, H_p, was taken as 0.6 and the length, L_{fbm}, was set to N (MATLAB version 2017a, Wavelet Toolbox). The Reconstructed trajectories and original DCTCs are given in Figures 2.13 - 2.16. The *x*-axis of the plot of DCTCs is in

logarithmic scale to provide a better resolution for the early DCTCs. The corresponding reconstruction error norms are given in Tables 2.8 - 2.11.

The compression power of DCT worked efficiently on these larger data sets. Figures show that *Ndctc*=10 yielded approximately reconstructed smooth trajectories even for N=100000. When *Ndctc*=20, the noise content of the original signal became more clear after reconstruction. Furthermore, almost all major fluctuations of the FBM curve were reconstructed by *Ndctc*=50. It should be noted that when N=100000, reconstruction with *Ndctc*=50 required 999950 zero elements and its reconstructed trajectory is very similar to the original data set. In other words, the success of the parameterization of a complex (random, non-repeating) trajectory of 100000 samples with barely 50 DCTCs prove that the method suggested in this thesis work, i.e., the CVP-DCT is a viable technique for compressive parameterization of optimal control trajectories of unknown form.

Table 2.8. Error norms in reconstruction of the FBM Signal with N=100 and H_p=0.6.

Ndctc	5	10	15	20	30	50
Error	0.0467	0.0318	0.0269	0.0235	0.0171	0.0114

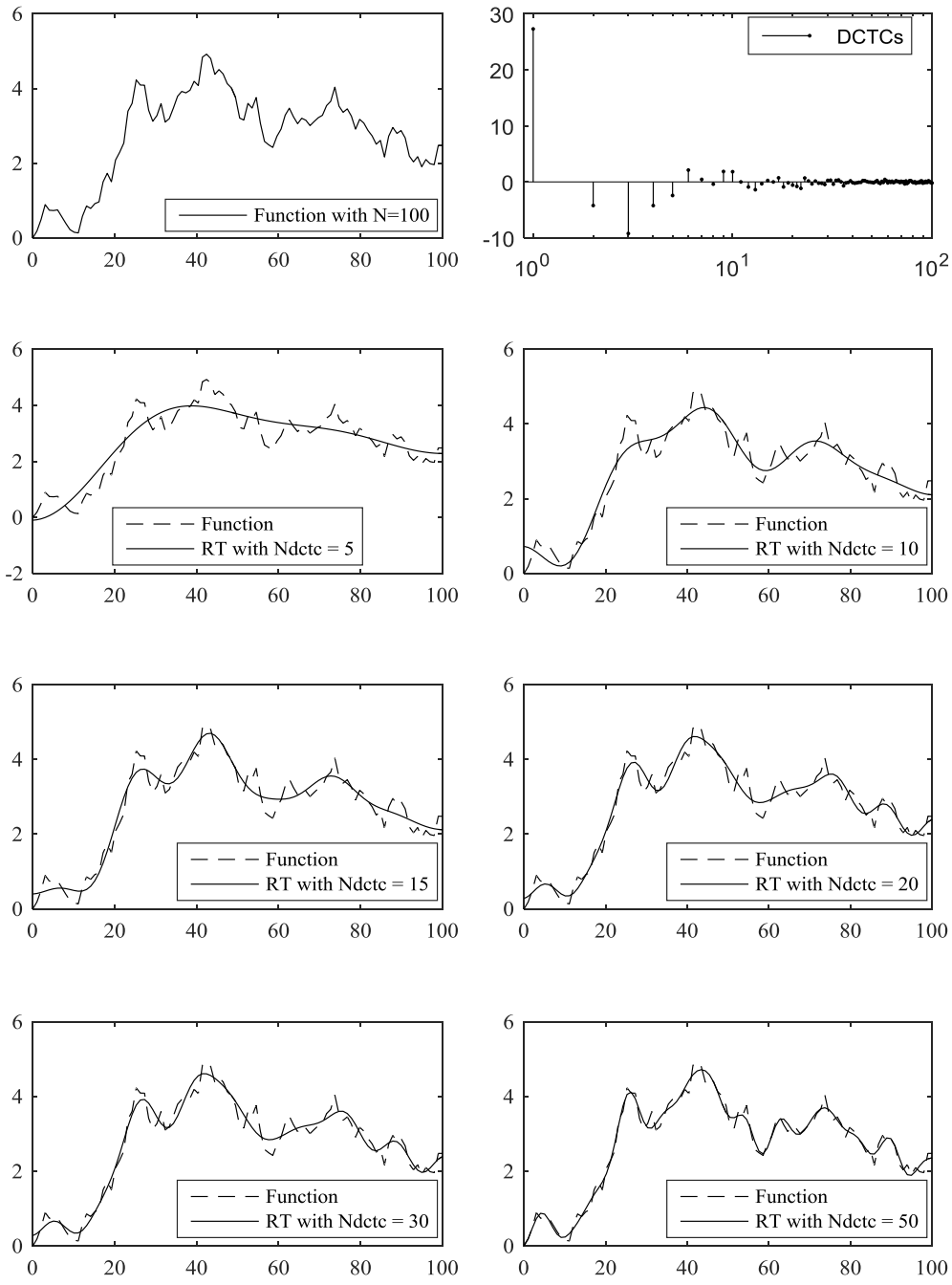


Figure 2.13. A FBM signal sampled with N=100 and H_p=0.6, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

Table 2.9. Error norms in reconstruction of an FBM Signal with N=1000 and H_p=0.6.

Ndctc	5	10	15	20	30	50
Error	0.0784	0.0595	0.0353	0.0311	0.0197	0.0155

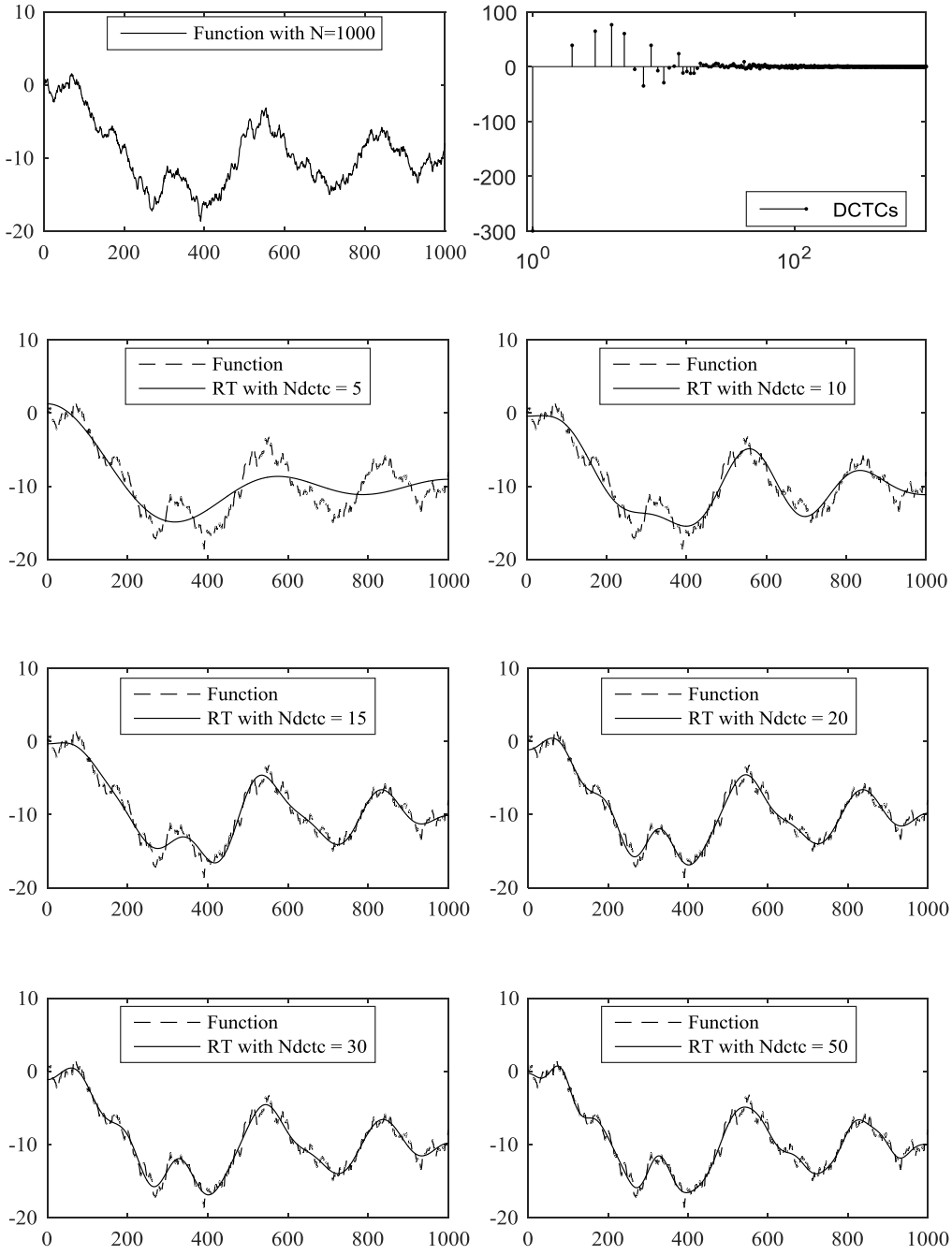


Figure 2.14. A FBM signal sampled with N=1000 and H_p=0.6, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

Table 2.10. Error norms in reconstruction of the FBM Signal with N=10000 and H_p=0.6.

Ndctc	5	10	15	20	30	50
Error	0.0921	0.0717	0.0325	0.0319	0.0296	0.0199

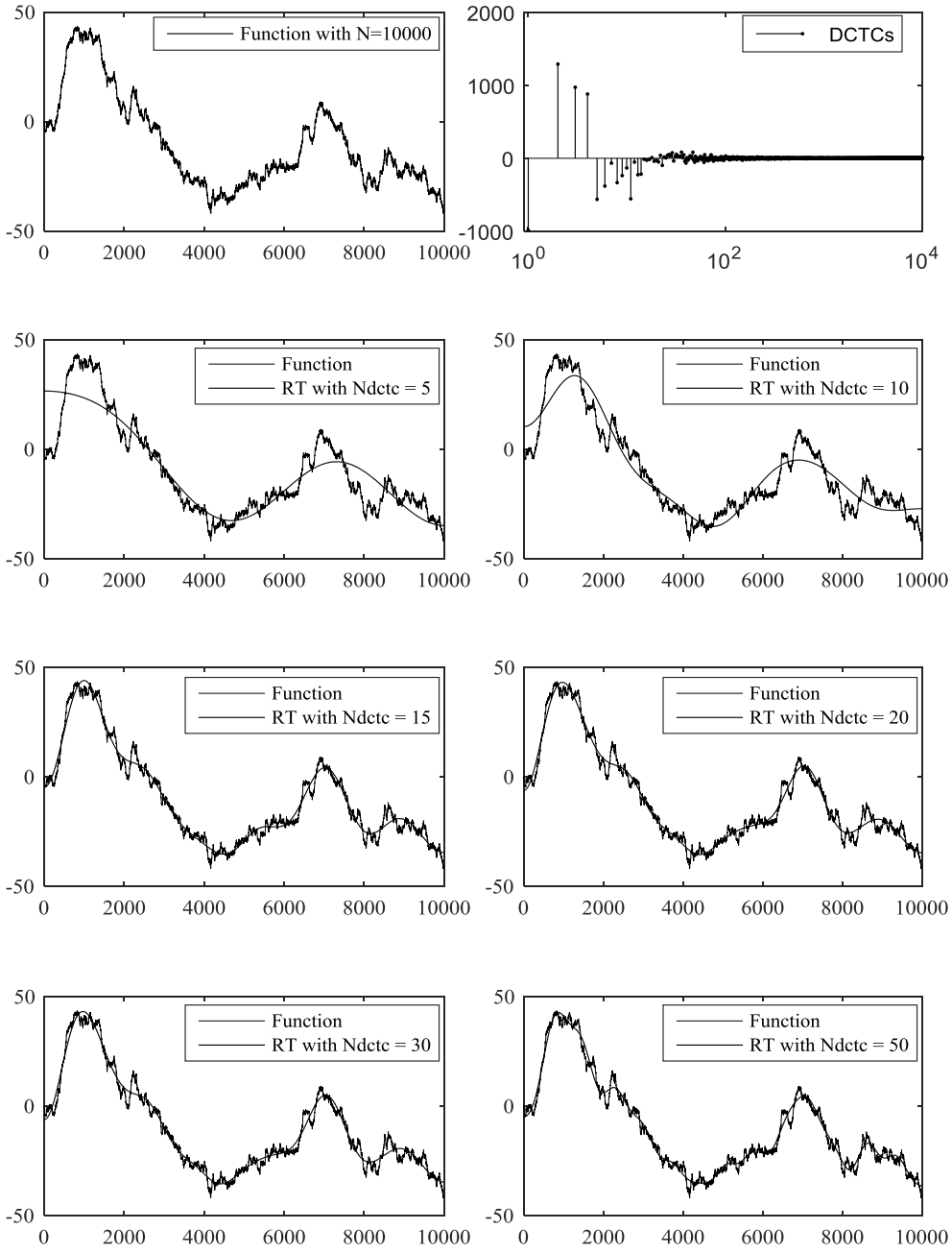


Figure 2.15. A FBM signal sampled with N=10000 and H_p=0.6, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

Table 2.11. Error norms in reconstruction of the FBM Signal with N=100000 and H_p=0.6.

Ndctc	5	10	15	20	30	50
Error	0.1081	0.0756	0.0502	0.0473	0.0394	0.0265

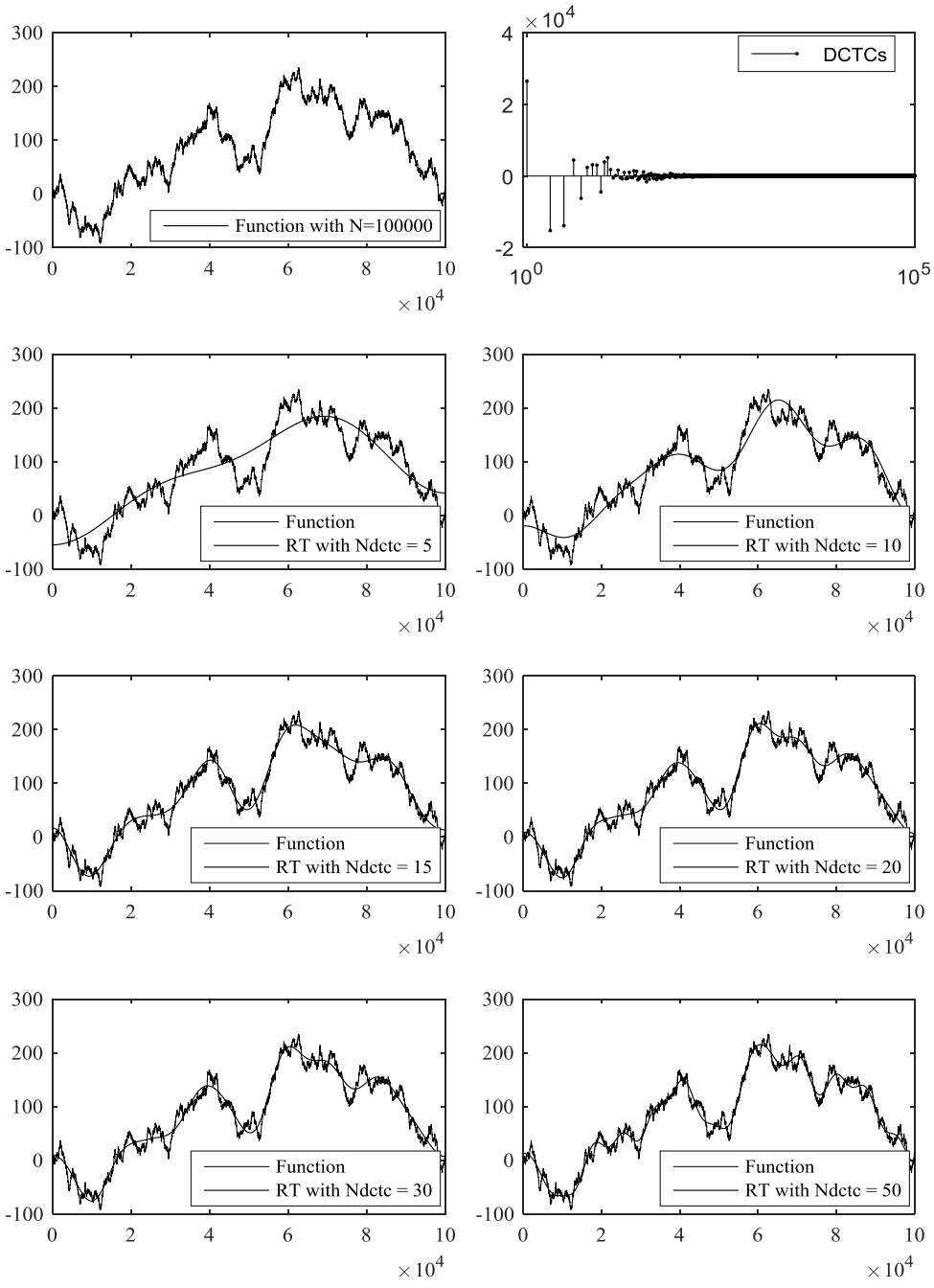


Figure 2.16. A FBM signal sampled with N=100000 and H_p=0.6, its DCTCs, and reconstructed trajectories for different *Ndctc* values.

To sum up, in this chapter, we mentioned the general properties of DCT and we focused specifically on its compression power. In sections 2.1.1-2.1.6, we examined simple functional forms, and in Section 2.1.7 we investigated the compression of lengthy and complex random FBM signal. In all cases, we saw that DCT is a very powerful data compression, and thus low-dimensional function parameterization technique. In fact, any trajectory with arbitrary shape and arbitrary complexity can be represented almost exactly or approximately in the DCT domain (domain of DCT coefficients) with only few parameters (DCTCs) compared to the length of the original trajectory. We envisioned the sample trajectories as optimal control trajectories, and this is the point from which the idea of this thesis, i.e. the Compressive Control Vector Parameterization, has emerged. The control vector parameterization with DCTCs is discussed in Chapter 5.

3. OPTIMAL-CONTROL THEORY

The history of Optimal-Control Theory (OCT) can be divided into two eras: early times (from the 17th century to 1950s) and modern times (1950s to today). The history of OCT began with the emergence of Calculus of Variations (CV) in the 17th century. Therefore, OCT is considered as an extension of CV. Notable contributors (or the ancestors) of OCT and CV are Fermat and Galilei in the 17th century; Bernoulli, Euler, and Lagrange in the 18th century; and Legendre, Jacobi, and Hamilton in the 19th century. Goldstine (1980) and Sussmann (1997) published two works in which they mention the scholarly history of CV and OCT. The modern era of OCT has started after the World War II. The competition between USA and USSR in the aerospace field (‘The Space Race') caused notable developments in OCT and its applications. This competition has left a great legacy for science. Bellman’s Dynamic Programming and Pontryagin’s Maximum Principle (PMP) were proposed in those years Bryson (1996) and Pesch (2012) published articles which include the detailed history of OCT after the 1950s. Today, optimal-control is an important tool for not only in science and engineering but also in economics, finance, and other social sciences.

Essentially, optimal-control is the determination of the best time variant control strategy for a dynamic system. The term “best” refers to the desired performance criterion (objective functional) and encompasses the constraints under dynamic conditions.

Optimal-control problems consist of three components. The first element is a mathematical description (model) of the system. The model describes the physical plant by a set of nonlinear or linear differential equations (mostly ODEs) The second component is the performance index (objective functional) which is aimed to be minimized or maximized. The term ‘functional’ is used to signify that the performance index is a function of the function of control(s) which, itself, is time dependent. Performance index is a function of control inputs, state variables and time. For example, in chemical engineering problems, performance index can be related to the batch time, feed flow rate, composition of a compound, energy supplied to the system, etc. The third component consists of the physical

constraints and boundary conditions of state and control variables. These constraints can be set as inequality or equality conditions and thus are called “inequality or equality constraints”. The constraints that are set only at the final time are called “terminal constraints” or “final-time constraints”. If the constraints are specified throughout the operation time (between the initial and final times) they are called as the “path constraints”; e.g., “inequality/equality path constraints”.

In this chapter, general formulations and solution methods of optimal-control problems are provided.

3.1. General Formulations of Optimal-Control Problems

Optimal-control problems for fixed final-time and continuous systems are formulated as follows (Kirk, 1970; Bryson and Ho, 1975; Lewis, 2012).

Performance index, J, to be minimized, defined in Bolza form, is

$$J = \Phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \tag{3.1}$$

The $\Phi(\cdot)$ term represents the final-time cost function and the $L(\cdot)$ term represents time dependent cost functional (function of states and controls that are functions of time). The independent variable is time t. $\mathbf{x}(t) \in R^n$ are n-dimensional vector of time dependent state variables and $\mathbf{u}(t) \in R^m$ represent m-dimensional vector of control variables. When only the integral term exists in the performance index, in other words, if the $\Phi(\cdot)$ term is not present, the performance index is referred to be in the Lagrange form which is given by (3.2). On the other hand, if the performance index is based only on the $\Phi(\cdot)$ term, in other words, if the $L(\cdot)$ term is not present, the performance index is referred to be in the Mayer form. Note that all the forms are mathematically equivalent but not comparable (Betts, 2010).

The Lagrange form,

$$J = \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \tag{3.2}$$

The Mayer form,

$$J = \Phi(\mathbf{x}(t_f), t_f) \tag{3.3}$$

The performance index is evaluated with respect to the system model, or in other words, the plant equations, which are typically in the form of ODEs with associated initial conditions.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{3.4}$$

$$\mathbf{x}(t_0) = \mathbf{x}(0) \tag{3.5}$$

The optimal-control problem is to find the optimal input vector, $\mathbf{u}^*(t)$, over the $[t_0\ t_f]$ time interval that drives the plant (3.4), starting from the initial states (3.5), along with an optimal trajectory $\mathbf{x}^*(t)$ such that the performance index is minimized.

The inequality path constraints (**C**) and equality path constraints **(E)** are defined as,

$$\mathbf{C}(\mathbf{x}(t), \mathbf{u}(t), t) \leq \mathbf{0} \tag{3.6}$$

$$\mathbf{E}(\mathbf{x}(t), \mathbf{u}(t), t) = \mathbf{0} \tag{3.7}$$

Terminal (final-time) inequality conditions (**Ω**) and terminal equality constraints (**Ψ**) are defined as follows.

$$\mathbf{\Omega}(\mathbf{x}(t_f), t_f) \leq \mathbf{0} \tag{3.8}$$

$$\boldsymbol{\Psi}(\mathbf{x}(t_f), t_f) = \mathbf{0} \tag{3.9}$$

This formulation (3.1-3.9) is also named as the "direct formulation". On the other hand, indirect formulation is also widely used. The key element of the indirect formulation is the presence of a scalar function called the "Hamiltonian", denoted by H. The followings can be derived by applying the theory of calculus of variations to the objective functional together with the state equations and constraints (Equations 3.1 to 3.9).

$$H(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), t) = L(\mathbf{x}(t), \mathbf{u}(t), t) + \boldsymbol{\lambda}^{\mathbf{T}}(t)\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{3.10}$$

The vector of adjoint variables (costates) is denoted by $\boldsymbol{\lambda}(t) \in R^n$. State equations that given in Equation (3.4) are defined as the partial derivative of Hamiltonian with respect to costate variables. State variables are integrated forward starting from the specified initial conditions.

$$\dot{\mathbf{x}}(t) = \frac{\partial H}{\partial \boldsymbol{\lambda}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \qquad t \geq t_0 \tag{3.11}$$

The costate variables are defined as the partial derivative of the Hamiltonian with respect to state variables and they are to be integrated backward starting from their final-time conditions.

$$-\dot{\boldsymbol{\lambda}}(t) = \frac{\partial H}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}^{\mathbf{T}}}{\partial \mathbf{x}}\boldsymbol{\lambda} + \frac{\partial \mathbf{L}}{\partial \mathbf{x}}, \qquad t \leq t_f \tag{3.12}$$

The stationarity condition (necessary condition of optimality) is defined by taking the partial derivative of the Hamiltonian with respect to the control variables. The stationarity condition leads us to express control variables in terms of state and costate variables. Stationarity condition is given in Equation (3.13).

$$H_u = \frac{\partial H}{\partial \mathbf{u}} = \frac{\partial \mathbf{f}^T}{\partial \mathbf{u}} \boldsymbol{\lambda} + \frac{\partial \mathbf{L}}{\partial \mathbf{u}} = 0 \tag{3.13}$$

The boundary conditions (transversality conditions) can be derived from the CV as well and are defined as follows.

$$(\boldsymbol{\Phi}_x + \boldsymbol{\Psi}_x^T \boldsymbol{\nu} - \boldsymbol{\lambda})^T|_{t_f} dx(t_f) + (\boldsymbol{\Phi}_t + \boldsymbol{\Psi}_t^T \boldsymbol{\nu} + H)|_{t_f} dt_f = 0 \tag{3.14}$$

where $\boldsymbol{\nu} \in R^p$ represents a vector of Lagrange multipliers that are static (time independent).

Furthermore, OCP can also be formulated by Hamiltonian-Jacobi-Bellman (HJB) equation (Bellman, 1966; Kirk, 1970; Luus, 2000) (or Dynamic Programming approach). However, this method, which is scarcely used especially in the applied optimal-control literature, is neither used nor discussed in this thesis work.

In the following sections, some basic solution methods of optimal-control problems are discussed.

3.2. The Analytical Solution Method

In a typical analytical solution procedure, the state ODEs are analytically solved using the initial conditions. At this stage, the stationarity condition is used to eliminate the controls in terms of states and/or costates. Using the closed form analytical solution of the state ODEs and the control(s) eliminated from the stationarity condition, the costate ODEs are analytically solved using the final-time conditions that can be derived by Equation (3.13). Since the costate ODEs are state dependent, the analytical solution of costates can be much more difficult than that of the ODEs. The solution of the states and costates are used to find an explicit form of the controls, i.e. as a function of time only. As can be understood, this procedure is rather limited and can be applied only to simple and mostly linear systems. The key issue is to be able solve the stationarity condition for the control(s) explicitly, otherwise

the rest of the procedure becomes too cumbersome or even impossible to carry out. Therefore, the analytical solution of nonlinear optimal-control problems are not available in general, and numerical solutions are widely preferred (Subchan, 2009). Since, the chemical engineering problems usually have nonlinear terms (e.g., reaction kinetics terms involving the exponential Arrhenius expression), it is not practical to solve them analytically, and is typically impossible to do so. Analytical solutions have many obstacles and drawbacks. When the number of states is higher ($n \geq 2$), satisfying the necessary condition of optimality requires considerable amount of symbolic computation (memory and CPU time). The fundamentals and limitations of the analytical approach are discussed by (Srinivasan *et al*., 2003). In this thesis work, analytical solutions are not investigated. However, a simple example problem is solved and compared with the numerical solutions is in Section 6.1.

Moreover, by taking total time derivative of the Hamiltonian following equation can be derived.

$$\dot{H} = \frac{\partial H}{\partial t} = H_{\mathbf{t}} + \mathbf{H}_{\mathbf{u}}^{T}\dot{u} + \left(\mathbf{H}_{\mathbf{x}} + \dot{\lambda}\right)^{T}\mathbf{f} \tag{3.15}$$

If $\mathbf{u}(t)$ is the optimal control input, $\mathbf{u}^{*}(t)$, then Equation (3.15) becomes,

$$\dot{H}(t) = H_{t} \tag{3.16}$$

If $\mathbf{L}(\cdot)$ and $\mathbf{f}(\cdot)$ terms do not have explicit time dependence, then the Hamiltonian is not time dependent either. In other words, the Hamiltonian is constant along the optimal trajectory. Therefore, total time derivative of the Hamiltonian is zero. This fact implies that, for explicitly time-independent cases, the Hamiltonian is zero over the optimal path.

$$\dot{H}(t) = 0 \tag{3.17}$$

3.3. Numerical Solution Methods

Numerical solution methods are classified into two categories; direct methods and indirect methods. Direct methods compute the problem (3.1-3.9) by discretizing state and control variables (simultaneous approach), or by discretizing only control variables (sequential approach) to create a Nonlinear Programming Problem (NLP). The NLP problem established after discretization of either the controls only or both the controls and the states can then be solved by various optimization methods (Bertsekas, 1999; Biegler, 2010). Indirect methods follow the Hamiltonian system, and thus utilizes the costate information as well, as described briefly in the previous section.

3.3.1. Indirect Numerical Methods

According to PMP, the optimum solution is found by minimizing the Hamiltonian, which is not straightforward in its implementation and requires various numerical procedures (Luus, 2008). The indirect methods deal with the obstacles of minimizing the Hamiltonian by iterative procedures. This approach uses the formulation (3.10-3.14) and converts the optimal-control problem to a 2-n dimensional Two Point Boundary Value Problem (TPBVP). The TPBVP can be handled via different methods such as the shooting, multiple shooting, and collocation methods (Rao, 2009).

The shooting method is also referred to as the Boundary Condition Iteration (BCI) (Luus, 2001; Paulen and Fikar, 2016). The state and costate equations (defined through Hamiltonian) are both integrated forward. Initial conditions of states are known and the missing initial conditions of the costates are guessed. This method aims to satisfy terminal conditions of the costates by adjusting their missing initial conditions. After integration, terminal conditions are compared to the boundary conditions and repeated until the difference is smaller than the error tolerance. If the controls can be solved explicitly in terms of state and/or costates from the stationarity condition, the procedure boils down merely to the solution of a TPBVP problem. However, if it is not possible to eliminate u explicitly,

then an optimization or set of nonlinear algebraic equation solution procedure, in which the controls at the prespecified time grids become the decision variables, is unavoidable. Moreover, the same procedure is followed by dividing the time interval into subintervals and solving the TPBVP on each interval, also by imposing the continuity conditions at the junctions of the consecutive subintervals. This method is called the Indirect Multiple Shooting Method (Rao, 2009; von Stryk, 1992)

The gradient method is also named as the Control Vector Iteration (CVI) (Srinivasan *et al*., 2003). Control input is discretized and the optimal control trajectory is searched by a suitable optimization algorithm along the gradient direction obtained through Equation (3.13). The 1st-order gradient method is the most basic one and in this method, the controls at each time grid are updated along the steepest-descent direction (Kirk, 2004). CVI is applied using following formula.

$$\delta \boldsymbol{u} = -\varepsilon \frac{\partial \mathrm{H}}{\partial \mathbf{u}} \tag{3.18}$$

where δ represents the variation of control variables and ε represents the step size (line-search parameter) which depends on the optimization strategy. Iterations through Equation (3.18) are performed until Equation (3.13) is satisfied.

The third indirect method is called indirect collocation. The state and costate are parameterized using orthogonal, and possibly with piecewise, polynomials and then discretized on a finite number of time grids via the collocation method (usually called the orthogonal collocation) (Srinivasan *et al*., 2003) The collocation method converts the solution of the state and costate ODEs (or the TPBVP) to the solution of large set of nonlinear algebraic equations (often called the residuals). The stationarity condition Equation (3.13) is added to this system of algebraic equations. These equations are then cast as the equality constraints of the optimization problem, the objective function of which is the performance index. The method results in very large NLP problem where, the controls, states, and possibly the costates are the decision variables. The key issue in these collocation-based

methods is to be able to give the objective and constraint gradients to the optimizer as accurately and efficiently as possible.

3.3.2. Direct Numerical Methods

Solving optimal-control problems via direct methods are done by applying the general optimal-control formulation given as Equation (3.1-3.9). The optimal-control problem is transcribed into an NLP problem and it is solved by NLP algorithms. The most commonly used direct methods are divided into two main groups which are "simultaneous" and "sequential" methods. This classification is based on parameterization (discretization) strategy applied. In simultaneous methods, both the state and control variables are parameterized/discretized (also called "complete parameterization" or "direct transcription"). For the discretization, various techniques can be utilized, such as orthogonal collocation, pseudospectral collocation etc. Whereas, in sequential methods (also called "feasible-path methods"), only the control variables are parameterized and state ODEs are integrated in every iteration of optimization.

The direct simultaneous methods are discussed in the next section. The direct sequential methods, on the other hand, are discussed in Chapter 4 separately, since this is the method used in this thesis work.

3.3.2.1. Simultaneous Direct Methods. In this method, the control variable(s) are discretized as in the sequential methods. However, this method does not use ODE integration on each step as done in the sequential methods, and the state variables are discretized as well. These methods convert the solution of the state ODEs to the solution of large set of nonlinear algebraic equations (often called the residual equations). These equations are then cast as the equality constraints of the optimization problem, the objective function of which is the performance index. The method results in a very large NLP problem where the controls and states at the discretization points become the optimization decision variables. The key issue in these simultaneous methods is to be able to give the objective and constraint gradients to

the optimizer as accurately and efficiently as possible. The resulting large-scale NLP problem usually requires special solution techniques (Biegler, 2007). Optimization is carried out in the full space of discretized inputs and states. The process model equations and constraints are satisfied simultaneously at the optimum point.

3.3.3. Summary of Numerical Methods

In the previous sections, an overview of numerical methods is provided. The scheme in Figure 3.1 represents the general classification of these methods.

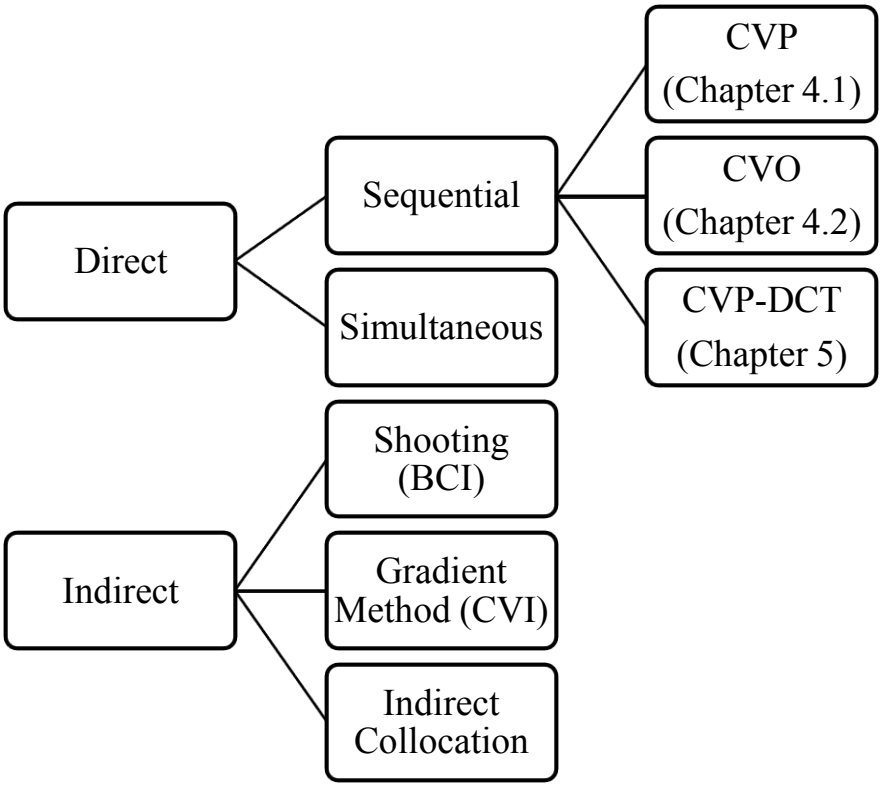


Figure 3.1. General classification of numerical methods for optimal-control problems.

4. DIRECT SEQUENTIAL METHODS

As stated in Chapter 3, the direct methods transform the infinite-dimensional optimal-control problems into finite-dimensional Nonlinear Programming (NLP) problems. This transcription is done by either parameterizing both the state and control variables (simultaneous method or complete parameterization) or parameterizing only the control variables; so-called the "direct sequential" methods.

In the direct sequential methods, the optimization is carried out in the space of input variables only. Control variables are approximated over $[\mathrm{t}_0, t_f]$ by a finite set of local or global functions. Therefore, the infinite-dimensional optimal-control problems are converted to finite-dimensional problems (Brusch and Schapelle, 1973; Goh and Teo, 1988; Gritsis, 1990; Kraft, 1985; Morison, 1984; Vassiliadis, 1993). In this method, the objective functional evaluations are carried out by the solution of the set of state ODEs given in Equation (3.4). The integration is implemented with standard Initial Value Problem (IVP) solvers (e.g., explicit or implicit methods such as the Euler, Runge-Kutta, backward-differentiation-formula based stiff integration methods, etc.) and the performance index J is evaluated. This method corresponds to a "feasible-path" approach since the differential equations are satisfied at each step of the optimization (Vassiliadis *et al.*, 1994). IVP integration tolerance, for local truncation-error control, is a critical parameter in the sequential method. If the lower-level ODE integration tolerance is close to or larger than the upper-level optimization-related tolerances (termination tolerance or derivative-evaluation tolerance) then resulting solution can be numerically unreliable. This may trigger premature termination of the optimization due to failure to find a correct search direction or may cause the optimizer to get trapped into false local minima (sensed by the optimizer due to loose ODE integration tolerances) (Banga *et al.*, 2003).

The sequential methods are relatively easy to implement; however, they require repeated numerical integrations at the lower level, from initial time to final time, at each iteration of the optimizer at the upper level of the algorithm. In other words, sequential methods are computationally expensive. Moreover, sequential methods are not guaranteed

to handle open-loop unstable systems. In the presence of instability, sequential methods may fail, and an alternative method may be needed (Biegler, 2010). Another drawback of the sequential methods is the possible non-convexity sensed by the optimizer. Non-convex and multimodal problems require robust NLP solvers because the solutions often end up with one of the "pseudo-minima"(von Stryk Bulirsch, 1992). To overcome these problems in this thesis work, it was preferred to implement an optimization sequence that starts with a Sequential Quadratic Programming (SQP) algorithm (derivative-based local solver) as typically suggested by many references (Canto *et al.*, 2002; Schlegel *et al.*, 2005) and then continues with the Differential Evolution (DE) algorithm (derivative-free global solver).

In this thesis work, the direct sequential method was categorized into two sections: Control Vector Parameterization (CVP) and Control Vector Optimization (CVO). This classification is based on the type of decision variables which are explained in following sections.

4.1. The Control Vector Parameterization Method

Control Vector Parameterization (CVP) is also called as the Direct Shooting Method (Rao, 2009). In this method, optimization is performed in the space of the parameters of control variables only. Control variables are approximated by functional forms. Optimization decision variables are the vector of functional parameters, **p**, the length of which is Np. In this method, a control variable is represented, e.g., as:

$$u(t) \cong f(\mathbf{p}, t) \tag{4.1}$$

where Np represents the total number of parameters.

For CVP, several functional forms were used in this thesis work. These functional forms are given in Table 4.1.

Table 4.1. Functional forms used in the CVP method in this thesis work.

Functional form	**Np**	$f(\mathbf{p_i}, t)$
(i) Constant + exponential function	3	$p_1 + p_2\ e^{p_3 t}$
(ii) Constant + exponential term	2	$p_1 + p_2\ e^{t}$
(iii) Constant + 1st order polynomial + exponential term	4	$p_1 + p_2 t + p_3\ e^{p_4 t}$
(iv) 2nd order polynomial	3	$p_1 + p_2 t + p_3 t^2$
(v) 3rd order polynomial	4	$p_1 + p_2 t + p_3 t^2 + p_4 t^3$
(vi) 4th order polynomial	5	$p_1 + p_2 t + p_3 t^2 + p_4 t^3 + p_5 t^4$
(vii) 5th order polynomial	6	$p_1 + p_2 t + p_3 t^2 + p_4 t^3 + p_5 t^4 + p_6 t^5$

The algorithm of the CVP method is given as follows:

(i) Select a functional form, i.e., parameterize the control.

(ii) Select an initial guess for the parameters.

(iii) Generate continuous control inputs over time, u(t), based on this parameterization.

(iv) Integrate the state ODEs and calculate the performance index.

(v) Repeat steps (iii) and (iv) within the optimization loop until the performance index is minimized or maximized.

Generally, good approximations are provided by this method, but local solutions are also possible. Hicks (1971) reported that different initial guesses might bring different solutions. Therefore, to prevent possible local solutions, either good (educated) initial guesses should be supplied, or a strong NLP solver with global search capabilities should be used. The CVP method implements globally parameterized functions, i.e., the functions are parameterized over the entire time horizon from initial time to final time. Thus, it may work acceptably even with only few number of parameters, especially if the control input is expected to be smooth and simple enough to be approximated with a readily known global function. A solution of complex control systems can easily be obtained even with low dimensional parameter estimation scheme. Generating continuous curves by higher order

polynomials is always possible, but it should be noted that higher number of parameters also means higher number of decision variables.

Since the 1970s this class of methods has been improved by replacing global parameterization with piecewise functions (Spangelo, 1994) which we called Control Vector Optimization and discussed in the next section.

4.2. The Control Vector Optimization Method

The Control Vector Optimization (CVO) method is also known as the Direct Multiple Shooting method. In this method, control input is parameterized explicitly by piecewise constant parameters. The states are calculated by the integration of state ODEs forward in time. If the final time (t_f) fixed, the time domain is divided into finite number of subintervals. In order to divide into k time stages, [k+1] time grid points must be used. The total number of time grids was denoted as *Ntgrid* in this thesis work. The grids have equal time lengths.

$$0 = t_1 < \ldots < t_k < \cdots < t_{Ntgrid} = t_f, \qquad k = 1, 2, \ldots, \ Ntgrid$$

$$\mathrm{t_{k+1}} - \mathrm{t_k} = \text{constant}$$

The optimum control variable on each time grid is a decision variable of the optimizer and their optimal values are found by an optimization algorithm. The control inputs, u(t), between the grid points are generated by piecewise (local) functions. In this thesis work we preferred to use interpolation methods. Therefore, the decision variables of the CVO method are the control variables on time grid points (u_k) and the number of decision variables is equal to *Ntgrid* directly as given in Figure 4.1.

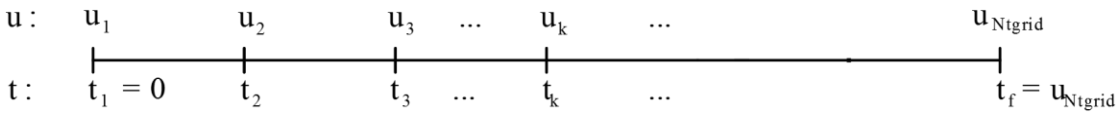


Figure 4.1. Discretization scheme of the CVO Method.

Therefore, the finite set of discretization parameters (decision variables) is given as

$$\mathbf{u}_{\text{grid}} = [u_1, u_2, u_3, \dots, u_k, \dots, u_{Ntgrid}]$$

The control inputs are mathematically formulated as follows.

$$\text{u(t)} \cong \text{g}\{u_{grid}(t_k),\ t\}, \qquad \text{k} = 1, 2, \dots,\ Ntgrid \tag{4.2}$$

where g{z, t} represents the interpolation of the grid quantity z to continuous time value, t.

In this thesis work, the "shape-preserving piecewise cubic interpolation (pchip)" and "cubic-spline interpolation (spline)" methods were used. Both methods are continuous, use local polynomials and require at least four grid points (MATLAB (version R2017a)). Thus, current optimization decision variables defined over the discrete grids, u_{grid}, are interpolated to any arbitrary continuous time, t, as required by the ODE integrators. Therefore, the method becomes more simple to implement and requires relatively less CPU time. The interpolations were directly implemented by the griddedinterpolant function of MATLAB with available "pchip" and "spline" options.

A comparison scheme of two methods is given in Figure 4.2. The "spline" produces smoother trajectories but it may exhibit overshoots. The "pchip" is not as smooth as "spline" but it has negligible overshoots for t that fall between the grid points t_k and t_{k+1}, and mostly behave very similar to linear interpolation. It should be noticed in Fig 4.2 that none of the controls at the grid points, $u(t_k)$, violates the upper and lower limits, $0 \le u(t) \le 1$. The pchip option does not cause violation of the limits in controls, u(t), for any arbitrary time t between

t_k and t_{k+1}, either. However, the spline option does cause significant violation of the control limits for times $t \neq t_k$.

Since the decision variables of this method are control variables on time grids, the interpolation method affects the optimization performance. Continuous (interpolated) control variables thus generated are used in integration of state ODEs, and therefore, affect the objective functional calculations.

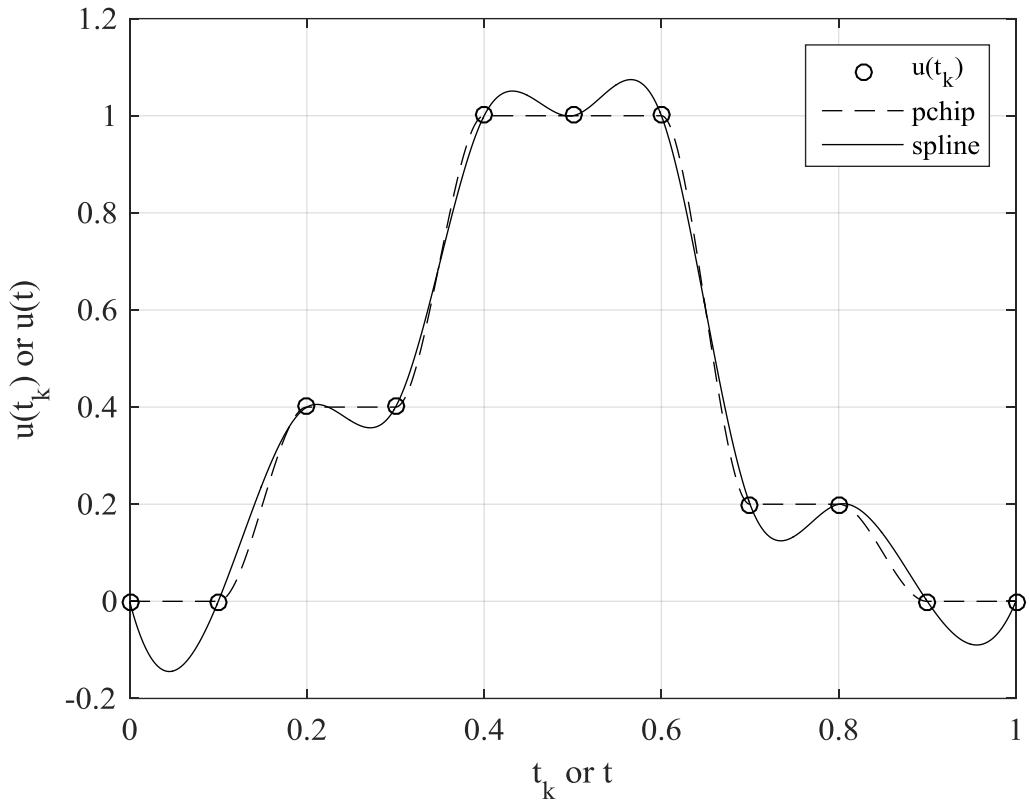


Figure 4.2. A comparison of the spline, and pchip interpolation methods.

(i) The algorithm of the CVO method is given as follows.

(ii) Select the number of time grids (*Ntgrid*) to divide time vector into *Ntgrid*–1 stages. Discretization scheme is given in Figure 4.1.

(iii) Create an initial guess vector $\mathbf{u}_{\text{grid}}$.

(iv) Integrate ODEs.

(v) During ODE integration generate continuous control input over time, u(t), by interpolation to integration time, t, through control points on each time grid.

(vi) Calculate the performance index.

(vii) Repeat steps (iii) to (v) within the optimization loop by creating new $\mathbf{u}_{\text{grid}}$ using the current values of the decision variables until the performance index is minimized or maximized.

Unlike the CVP method, the CVO method does not require a prespecification or assumption of the shape of the control trajectory, and thus, it can be applied to any problem without a priori knowledge. With fewer grid numbers there is the possibility of constraint violations between grid points (for equality and/or inequality path constraints). The use of higher number of grid points can lead to better solutions. However, the more grid points means the more decision variables and thus larger and more difficult NLP problem, more function evaluation, more CPU time requirements. Furthermore, increasing the dimension of the problem (more grids) does not guarantee a better solution (von Stryk, 1992). Since the interpolation method is applied trajectories may become smooth and continuous. However, in high-dimensional cases, control vector fluctuations are possible unless a smoothing function is applied. To conclude, the performance (J* and CPUs, etc.) of the methods are highly dependent to trajectory of the control vector. On one hand, the CVP may yield better solutions, more swiftly, requiring fewer parameters compared to CVO, however, identifying a functional form is the key issue. On the other hand, the CVO method can be applicable to any optimal-control problem regardless of the trajectory shape yet it may require larger CPUs, and usually, it ends up with a fluctuant control trajectory if higher number of decision variables are chosen and smoothing is not used. Therefore, as an attempt to remedy these problems, we ended up with a novel method called the CVP-DCT which is discussed in Chapter 5.

4.3. Handling Equality and Inequality Constraints in the Sequential Methods

Sequential methods are easy to implement in general, however, handling constraints usually causes some computational problems. There are several methods to handle the

constraints of an optimal-control problem. These methods were reviewed by Vassiliadis (1993) and Feehery (1998).

In this thesis work, the penalty-function method was used to handle constrained optimal-control problems. For constrained problems, penalty functions were added to performance indexes, consequently, problems were treated as unconstrained problems. The performance index is given in Equation (3.1) becomes as follows.

$$\mathrm{J} = \Phi(\mathbf{x}(\mathrm{t_f}), \mathrm{t_f}) + \int_{\mathrm{t_0}}^{\mathrm{t_f}} \mathrm{L}(\mathbf{x}(\mathrm{t}), \mathbf{u}(\mathrm{t}), \mathrm{t})\, \mathrm{dt} + \text{Penalty Function Terms} \tag{4.3}$$

In this method, the objective functional is penalized by adding the integral of constraint violations with a penalty weight parameter (Bryson and Ho, 1975; Upreti, 2012). The penalty functions can be applied to equality or inequality constraints given in Equations (3.6), (3.7), (3.8) and (3.9). The state inequality path constraints are handled with the Path-Constraint Penalty (PCP). The PCP function is defined as follows. Let $\mathbf{x}_{\boldsymbol{p}}(t)$ be the path limit as given in Equation (3.6) or the desired final time conditions given as Equation (3.8). Then:

$$\mathrm{PCP} = \sum_{\mathrm{k}=1}^{Ntgrid} \mu_{\mathrm{pc}} max\{0,\ \mathbf{x}(\mathrm{t}_k) - \mathbf{x}_{\mathrm{p}}(\mathrm{t}_{\mathrm{k}})\} \tag{4.4}$$

where μ_{pc} denotes the PCP parameter.

Inequality path constraints for control variables, e.g., upper and lower bounds of control variables, are handled by defining the Upper-Lower Bound Penalty Function as follows. Let a control bound is given with its lower bound (LB) and upper bound (UB).

$$\mathrm{LB} \leq \mathrm{u}(\mathrm{t}) \leq \mathrm{UB} \tag{4.5}$$

$$\mathrm{u_{min}} = min\{\mathrm{u}(t_k)\}, \quad \mathrm{k} = 1, 2, \dots,\ Ntgrid \tag{4.6}$$

$$\mathrm{u_{max}} = max\{\mathrm{u}(t_k)\}, \quad \mathrm{k} = 1, 2, \dots,\ Ntgrid \tag{4.7}$$

$$\mathrm{ULP} = \mu_{\mathrm{ul}}[max\{0,\ \mathrm{LB} - \mathrm{u}_{\min}\} + max\{0,\ \mathrm{u}_{\max} - \mathrm{UB}\}] \tag{4.8}$$

where μ_{ul} denotes the ULP parameter.

Final-time equality constraints, Equation (3.9), are handled by the Final-Time Equality Constraint Penalty (ECP). The ECP function is defined as follows. Let $\mathbf{x_T}$ be the desired final time conditions of the state variables. Then:

$$\mathrm{ECP} = \mu_{\mathrm{e}} \|\mathbf{x}(\mathrm{t_f}) - \mathbf{x}_{\mathrm{T}}\| \tag{4.9}$$

where μ_{e} denotes the ECP parameter. To sum up, the penalty function term becomes:

$$\text{Penalty Functions} = \mathrm{ECP} + \mathrm{ULP} + \mathrm{PCP} \tag{4.10}$$

Therefore, the constrained optimal-control problem becomes an unconstrained one. The penalty functions are quite simple to implement. However, this approach may be inadequate in some cases. State path constraints might be violated not only at the time grids but between the time grid points if the total number of grids is relatively low.

5. THE CVP-DCT METHOD

Previously, in Chapter 3, a brief overview of numerical methods for the solution of optimal-control problems had been mentioned and, in Chapter 4, two different direct sequential methods (CVP and CVO) had been discussed with their advantages and disadvantages. Considering those advantages and disadvantages we ended up with a novel method, called the Compressive Control-Vector Parameterization with Discrete Cosine Transform (CVP-DCT). In this chapter, some delicate features of the CVP-DCT method are provided, parameterization performance of DCT for CVP is discussed, and several possible control trajectories are examined.

The CVP-DCT method is also a sequential direct method which capitalizes on the compression power of DCT for the reconstruction of control inputs on time grid points. Therefore, we call it also as a "compressed parameterization" technique. Control vector $u(t_k)$, (k=1, 2, ..., *Ntgrid*) is discretized as it is done in the CVO method. However, this method takes the Discrete Cosine Transform Coefficients (DCTCs) of the yet-unknown (unspecified) control vector as decision variables. To be more specific, in the CVP-DCT method, the control inputs on time grid points are calculated by the inverse DCT (i-DCT) of a lower-dimensional (*Ndctc*) set of the decision variables (transmitted by the optimizer) that are zero-padded to *Ntgrid*. Since the values of any one of the DCTCs affect the reconstruction of the entire trajectory, this method corresponds to a "global parameterization method" as previously given in the CVP method (see Chapter 4.1). The DCT may be represented in matrix form as previously given in Equations (2.3) and (2.5). The CVP-DCT method in matrix form is given as follows.

$$\mathbf{c} = \mathbf{M}\ \mathbf{u} \tag{5.1}$$

$$\mathbf{u} = \mathbf{M^{-1}c} \tag{5.2}$$

$$\text{where} \begin{cases} \mathbf{u}\text{: discretized control vector } (Ntgrid \times 1) \\ \mathbf{M}\text{: discrete cosine transform matrix } (Ntgrid \times Ntgrid\ ) \\ \mathbf{c}\text{: discrete cosine transform coefficients } (Ntgrid\ \times 1) \end{cases}$$

c is a vector that consists of *Ntgrid* elements. Number of decision variables are the number of DCTCs (discrete cosine transform coefficients) (*Ndctc*) and the **c** vector includes [*Ntgrid*–*Ndctc*] (*Ntgrid* minus *Ndctc*) zero elements. Therefore, the optimization algorithm searches for the optimum DCTCs only, i.e., only the first *Ndctc* values of the *Ntgrid*-dimensional **c** vector. The algorithm of the CVP-DCT method is just an additional DCT parameterization section in the CVO's algorithm.

The algorithm of the CVP-DCT method is given as follows.

(i) Select the number of time grids (*Ntgrid*) to divide time vector into *Ntgrid*–1 stages. Discretization scheme is given in Figure 4.1.

(ii) Select number of DCTCs (*Ndctc*).

(iii) Set the initial guess for each DCTC (the first *Ndctc* elements of the vector **c**) and then complete the vector **c** to *Ntgrid* elements via zero-padding the remaining [*Ntgrid*–*Ndctc*] elements.

(iv) Create an initial guess vector $\mathbf{u}_{\text{grid}}$ by Equation (5.2), i.e., the i-DCT.

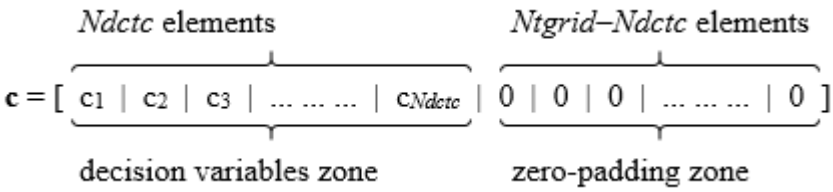
(v) Integrate ODEs.

(vi) During ODE integration generate continuous control input over time, u(t), by interpolation to current integration time, t, through control points on each time grid.

(vii) Calculate the performance index.

(viii) Repeat steps (iv) to (vi) within the optimization loop by creating new **c** vector using the current values of the decision variables as the first *Ndctc* elements of it and zero-padding the rest, until the performance index is minimized or maximized.

The construction of the decision-variable vector, **d**, for use in optimization can be summarized as follows. Taking the first *Ndctc* elements as the decision variables and zero-padding the rest up to *Ntgrid* elements creates the DCTCs vector **c**,



(5.3)

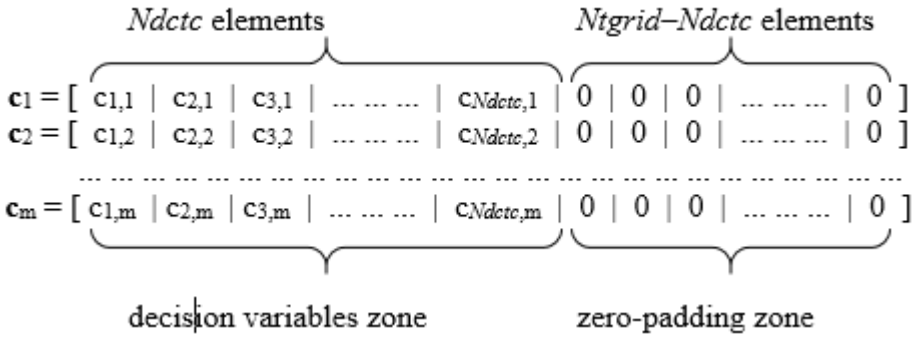
The vector of decision variables, **d**, perceived by the optimizer, is the first *Ndctc* elements of this DCTCs vector:

$$\mathbf{d} = [\ c_1 \mid c_2 \mid c_3 \mid \ldots\ \ldots\ \ldots \mid c_{Ndctc}\ ] \tag{5.4}$$

The current values of the controls, at the current iteration of the optimizer, on the time grids are obtained first by zero padding the Equation (5.4) (coming from the optimizer as decision variables) up to *Ntgrid* elements to form vector **c** and then taking the inverse DCT transform of **c** by Equation (2.5), yielding:

$$\mathbf{u} = [\ u_1 \mid u_2 \mid u_3 \mid \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \mid u_{Ntgrid}\ ] \tag{5.5}$$

This representation above is for the case when there is only one control input (m = 1). For the optimal-control problems that include more than one control, m > 1, the above construction scheme is repeated for each control independently.



(5.6)

However, in this case (for m > 1), the decision variables (the first *Ndctc* elements of the DCTCs of each input, $\mathbf{c}_1$ to $\mathbf{c}_m$) are concatenated to a single vector of decision variables, **d**, as follows. The total number of decision variables in this case is m×*Ndctc*.

$$\mathbf{d} = [\, c_{1,1} \mid \ldots \mid c_{Ndctc,1} \mid c_{1,2} \mid \ldots \mid c_{Ndctc,2} \mid \ldots \mid c_{1,m} \mid \ldots \mid c_{Ndctc,m} \,] \tag{5.7}$$

It should be noted that, there is no restriction to use the same or different *Ndctc* for each control input. Therefore, the total number of decision variables for the case where different *Ndctc* values are used for different control inputs, is

$$\text{Total number of decision variables} = \sum_{i=1}^{m} Ndctc_i \tag{5.8}$$

At a particular iteration of the optimizer, the current values of the controls over the time grids are obtained first by unconcatenating the **d** vector given in Equation (5.7), to the individual **c** vectors given in Equation (5.6), by zero padding of each individual **c** vector up to *Ntgrid* elements, and then by taking the inverse DCT transform of each **c** by Equation (5.2).

$$\begin{aligned} \mathbf{u}_1 &= [\, u_{1,1} \mid u_{2,1} \mid u_{3,1} \mid \ldots\ \ldots\ \ldots \mid u_{Ndctc,1} \,] \\ \mathbf{u}_2 &= [\, u_{1,2} \mid u_{2,2} \mid u_{3,2} \mid \ldots\ \ldots\ \ldots \mid u_{Ndctc,2} \,] \\ &\ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \\ \mathbf{u}_m &= [\, u_{1,m} \mid u_{2,m} \mid u_{3,m} \mid \ldots\ \ldots\ \ldots \mid u_{Ndctc,m} \,] \end{aligned} \tag{5.9}$$

The profile of the reconstructed trajectory strictly depends on *Ndctc* (see Chapter 2.1) At this point, recall the formulation of DCT given in Equation (2.1) and that the DCT is a summation of cosine functions with different parameters. When k=1, i.e., decompression with *Ndctc*=1, creates only a constant trajectory because cos(·) term is equal to zero. Decompression with *Ndctc=2* is summation of a constant profile and a cosine trajectory. Therefore, the CVP-DCT requires at least two DCTCs. Reconstructed control trajectories with *Ndctc* = 2 have very smooth profiles (similar to s-shaped curves) and they do not have local extrema (minimum or maximum points). However, the control trajectories reconstructed with *Ndctc*= 3 are smooth trajectories (similar to Gaussian shaped curves) as

well, yet these trajectories can have only one extrema. Similarly, trajectories with *Ndctc*= 4 can have two extrema. Therefore, we can conclude that when *Ndctc* ≥ 3, reconstructed control trajectories can have maximum [*Ndctc*−2] extrema. Thus, fewer *Ndctc* values (2 to 4) yields smoother trajectories and as *Ndctc* increases the possibility of oscillations in the reconstructed control trajectories increases. To check these features numerically, randomly created 10 vectors of DCTCs for each *Ndctc* from 2 to 10 on 101 time grids are represented in Figure 5.1. Basically, Figure 5.1 shows the possible samples of reconstructed control trajectories and gives information about what to expect from a selected *Ndctc* during an optimization.

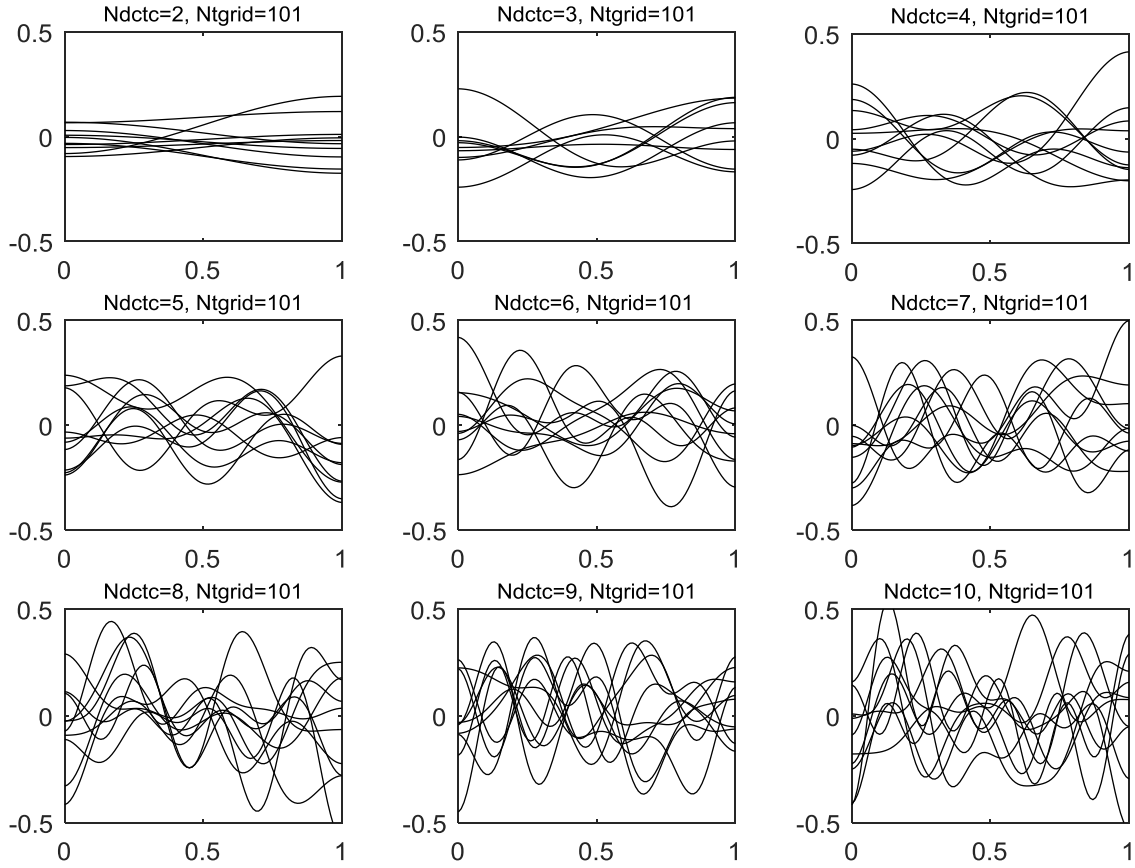


Figure 5.1. Possible samples of reconstructed control trajectories on *Ntgrid*=101 with different *Ndctc* values.

Since the norm of 2nd derivative (or, 2nd-order difference) is a measure of smoothness), the smoothness of the trajectories in Fig. 5.1 was checked numerically by

examining the norm of the 2nd-order differences of these randomly generated control trajectories as follows. For *Ntgrid*=101, *Ndctc* is selected a value between 2 and 10, and 1000 DCTC vectors with *Ndctc* random elements are created. Inverse-DCT of zero padded vectors decompresses to 1000 control vectors corresponding to these random DCTCs. The norm of the 2nd-order differences of the elements of those control vectors are calculated. The maximum of the norms calculated from 1000 samples are given in Table 5.1.

Table 5.1. Comparison of the maximum smoothness measures of 1000 control trajectories with different *Ndctc* for *Ntgrid*=101.

Ndctc	**Max norm of 2nd-order differences**
2	~0
3	0.004
4	0.009
5	0.018
6	0.029
7	0.044
8	0.061
9	0.082
10	0.109

Table 5.1 proves that with fewer number of *Ndctc*, the control vectors have smoother profiles, whereas those with higher *Ndctc* can have fluctuant, multimodal profiles. One worth mentioning point that Table 5.1 does not imply is that a trajectory obtained by *Ndctc* = 10 is always fluctuant and it has always 8 extrema points. If the optimizer sets the later DCTCs (those towards the end of the vector) to quite lower values, these fluctuations are not observed. To sum up these facts, we can conclude that parameterization with fewer *Ndctc* can provide a good smooth approximate trajectory even over very large time grids e.g. *Ntgrid*=1000, 10000, 10000. These low-order DCT parameterization approach yields good initial-guess trajectories for other numerical optimal-control methods. On the other hand, higher *Ndctc* has the potential to create more complex trajectories, however, at the expense

of possibly fluctuant trajectories that can make the optimization more difficult. Therefore, CVP-DCT method requires a strong NLP solver especially for high *Ndctc* values.

The CVP-DCT method can be applicable to any functional form, because it is free of the hassle of functional form specification, therefore it is more flexible than the CVP method. On the other hand, theoretically, the CVP-DCT method cannot find a better solution than the CVO method. Because the main job of the CVP-DCT method is to provide control inputs on time grids using fewer number of decision variables. However, in practice, the CVP-DCT method can provide better solutions on high *Ntgrid*. Especially on large time grids, finding the global solution by the CVO method may require over billions of function evaluations that may require days of computing. Since the CVP-DCT method decreases the number of decision variables it works much faster than the CVO method and it requires much fewer number of functional evaluations. Furthermore, in some extreme cases, CVP-DCT can find the global optimum with much fewer parameters.

6. EXAMPLE PROBLEMS SOLVED VIA THE CVP-DCT METHOD

In this chapter, several optimal-control problems are solved via the Compressive-Control Vector Parameterization with Discrete Cosine Transform (CVP-DCT) with different number of time grids, *Ntgrid*. Comparison of the optimal performance index values, state and control trajectories are also provided. For the first example, Problem 1 in this thesis work, performances of methods discussed in Chapters 3-5, i.e., analytical solution, indirect (TPBVP) solution, Control Vector Optimization (CVO), Control Vector Parameterization (CVP) and CVP-DCT are tested.

For the sequential solutions (CVO, CVP, and CVP-DCT), the following optimization task sequence is used. Firstly, for the "fmincon" function of MATLAB, which uses constrained nonlinear optimization algorithms such as the Sequential Quadratic Programming (SQP), Active Set Strategy, and Interior Point method (MATLAB version 2017a, Optimization Toolbox), the SQP algorithm is chosen. The optimization options of "number of maximum iterations", and "maximum number of function evaluations" are set as 5000 and 500, respectively. "termination tolerance of objective function" (*TolFun)* is set as 10^{-10}, "step-size tolerance" (*TolX)* are set as 10^{-8} that means if the norm of the difference of decision variables at consecutive steps is less than the set limit, the optimizer terminates. Rest of the options are left at their default values. "fmincon" is a gradient-based algorithm, and thus, it tends to stop on a local optimum. Therefore, the solution found by "fmincon" is assumed to be a local suboptimal solution and this local solution is introduced as the initial guess to another optimizer which uses Differential Evolution (DE) algorithm.

The DE algorithm was introduced by Storn and Price (1995). It is a gradient free, population-based unconstrained optimization method, which solves optimization problems by generating evolved candidate solutions, similar to genetic algorithms (Price *et al*., 2005). The DE consist of three main steps which are mutation, crossover, and selection. For each target vector in the population, a mutant vector is generated (mutation) and the selected target

vector is mixed with the mutated vector (crossover) then the fittest (or a random) vector is selected. There are two main parameters that affect the performance of DE. The "crossover probability constant" (*CR*) ∈ [0,1] controls which and what fraction of the components of the population vector are mutated at each step (Mallipeddi and Suganthan, 2010). The crossover can be accomplished through either binomial or exponential scheme. The "amplification factor" (F) ∈ [0,2] determines the mutation step size and it affects the CPU time to reach an extremum point. The effectiveness of *CR* and *F* depends on the optimization problem. In this thesis work, CR = 0.9 and F = 0.8 are used as suggested for difficult multimodal optimization problems (Price and Storn, 1997). The "maximum-iteration limit" is set as 2500 and "functional-tolerance limit", which is based on the standard deviation of the values of objective functions corresponding to current population of the decision variables, are set as 10^{-8}. Based on practical experience, "population size" is suggested to be set to 3-10 times larger than the number of decision variables (Gämperle *et al*.,2002). In the following examples, the number of decision variables changes from 2 to 11. Therefore, the population size is taken as 20 in all runs. The initial guess is taken from "fmincon"'s solution and used as a member of the initial population. Optimization strategy of DE is selected as the DE/best/1/bin. This strategy selects the best vector as the base vector, then adds a single difference vector to create a new vector by binomial crossover scheme according to selected *CR* and *F* parameters. The basics, detailed explanation of parameters and strategies of *DE* are available in the works of Price *et al*., (2005), and the developments are gathered in Chakraborty (2008).

The main reason of using this optimization sequence is to decrease the objective function value fast (with less number of function evaluations and less number of ODE solutions) by "fmincon" to, possibly, a local minimum. Then it is aimed to look for the global optimum around "fmincon"'s solution by capitalizing on DE's capacity for exhaustive searching and locating possibly the global minimum even far from the initial guess supplied. It should also be noted that DE generates only the initial population strictly between the user supplied lower and upper bounds on decision variables, but it disregards these bounds if potentially better populations tend to move outside these limits. Performing the optimization only with the DE works as well, however it requires longer CPU times. Since the DE algorithm selects population members randomly, measured CPU times can be different at

each run. Therefore, CPU times are investigated only for the CVO method and only for the first example problem.

Since in the sequential direct methods the system ODEs are integrated at each iteration of the optimizer, the accuracy in integration of the ODEs influences the results of optimization. Therefore, the ODE integrator must be selected carefully. In this thesis work, the state ODEs are solved by "ode15s" function of MATLAB. "ode15s" is a stiff ODE integrator which implements backward differentiation formula (BDF) with variable order (1^{st} to 5^{th} order), using variable step size (MATLAB version 2017a.; Shampine *et al*., 2003). Other ODE integrators of MATLAB could have been selected as well but following factors are considered. Firstly, in the example problems solved, the stiffness of ODEs is not known, hence it is practical to use stiff solvers such as "ode15s" or "ode23s". Non-stiff solvers are still able to solve stiff problems however they require more CPU time if the problem is stiff. Second reason is the accuracy, among those stiff ODE solvers, "ode15s" uses the highest-order formula when it is necessary. "ode113" function of MATLAB uses higher order (1^{st} to12^{th}) than "ode15s", however it is not a stiff solver. Therefore, "ode15s" is selected as the default ODE solver in this thesis work. Moreover, the Jacobian matrix of the ODEs and its sparsity patterns are introduced analytically to increase solver efficiency. The "relative" (*RTol*) and "absolute" (ATol) tolerance limits for the local truncation errors are set as 10^{-10} and 10^{-8}, respectively.

6.1. Problem 1

This problem belongs to Kirk (2004), (Example 5.1-1, pp.198-202). In order to compare, the problem is solved both analytically (using the Symbolic Math Toolbox of MATLAB), and numerically by the TPBVP, CVO, CVP and CVP-DCT methods. The problem is formulated as follows.

Given state ODEs.

$$\dot{x}_1 = x_2 \tag{6.1}$$

$$\dot{x}_2 = -x_2 + u \tag{6.2}$$

With the initial and final conditions specified as:

$$\mathbf{x}(0) = \mathbf{0} \tag{6.3}$$

$$\mathbf{x}(t_f) = \begin{bmatrix} 5 \\ 2 \end{bmatrix} \tag{6.4}$$

Given that $t_f = 2$, find the optimal control trajectory, $\mathbf{u}^*(t)$ that minimizes the following performance index,

$$J = \frac{1}{2}\int_{t_0}^{t_f} u^2(t)dt \tag{6.5}$$

Instead of post-computing the performance index (6.5) using numerical integration, e.g., by the trapezoidal rule, J is computed by taking the integrand as the right-hand-side function of an auxiliary state's derivative. In other words, an additional state ODE is adjoined to the system model with zero initial condition to integrate the integral term of the performance index simultaneously with the integration of the state ODEs. So, the performance index was converted from Lagrange form (3.2) to Mayer form (3.3). Even though both formulations are mathematically equivalent (Betts, 2010), this conversion was implemented to prevent possible numerical errors caused by evaluation of the integral separately. The auxiliary state ODE is defined as follows with zero initial condition.

$$\dot{x}_3 = \frac{1}{2}u(t)^2 \tag{6.6}$$

In this thesis work, the equality and inequality constraints are not handled directly, in other words, constraint function of the optimization algorithm ""fmincon"" is not utilized. Instead, the penalty-function technique is adopted. The fixed final-time condition (6.1.4) is adjoined to the performance index via the Equality Constraint Penalty (ECP) function as the Euclidean norm of the difference between the final-time states $x_1(t_f)$, $x_2(t_f)$ and the fixed final-time conditions given in Equation (6.7), where μ_e represents the penalty parameter for the equality constraints.

$$\mathrm{ECP} = \mu_e \left\| \begin{bmatrix} x_1(t_f) \\ x_2(t_f) \end{bmatrix} - \begin{bmatrix} 5 \\ 2 \end{bmatrix} \right\| \tag{6.7}$$

Considering the auxiliary state ODE, the performance index was re-defined as follows

$$J = x_3(t_f) + \mathrm{ECP} \tag{6.8}$$

6.1.1. Solution of Problem 1 analytically and numerically as a TPBVP

Problem 1 is solved analytically by following the theoretical optimal control solution that involves the Hamiltonian, state and co-state ODEs and the associated boundary conditions, as discussed in Chapter 3 and as formulated by Equations (3.6-3.12). The analytical solution is obtained with the help of MATLAB's Symbolic Math Toolbox. On the other hand, the indirect numerical solution is obtained by solving state and costate ODEs as a Two Point Boundary Value Problem (TPBVP) using MATLAB's "bvp4c" function which is based on collocation method as discussed in Chapter 3.3.1. Both methods give the same optimum value of the performance index, J*=16.7507, identical to four digits after the decimal. The analytic expression of optimum control input, u*(t), using necessary condition of optimality Equation (3.13). is given in Equation (6.9).

$$u^*(t) = \left(\frac{3\,e^2}{4}\right) - \frac{(3e^2 - 7)e^t}{2\,(e^2 - 1)} + \frac{7}{4} \cong 7.2918 - 1.187e^t \tag{6.9}$$

The corresponding **u***(t) and **x***(t) trajectories are shown in Figure 6.1. Since it is not possible to discriminate the optimal trajectories obtained by both methods, trajectories given on Figure 6.1 belong to the analytical solution.

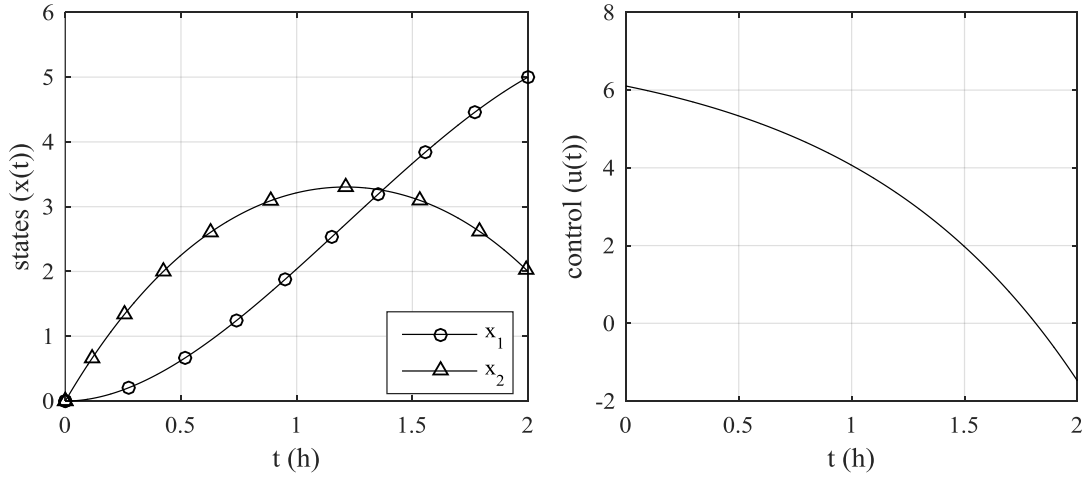


Figure 6.1. Optimal state and control trajectories of Problem 1 by the analytical and TPBVP solutions (J*=16.7507).

6.1.2. Solution of Problem 1 via the CVO Method

Problem 1 is solved via the CVO method on 11, 21, 31, 41, 51, and 101 time grids, *Ntgrid*. The calculated optimal performance index values, J*, are given on Table 6.1. The initial guess were given $u(t_k) = 0$, k=1, ..., *Ntgrid* for all runs. Optimization processes are terminated if the optimizer cannot find the global solution in 2500 iterations. Even though solving the problem on more time grids was expected to increase the probability of obtaining the global solution, the following results showed that solving the problem on more grids does not necessarily guarantee the global solution. Increasing the number of grids required more CPU time as shown in Table 6.1, because solving the problem on more *Ntgrid* creates larger NLP problems. It should be noted that the DE algorithm was used, and thus the CPU time results were not unique and were not reproducible as stated at the beginning of this chapter, however, they are given just to make a qualitative comparison. Increased CPU time was caused by the increased number of optimizer iterations. More grids, *Ntgrid*, means more decision variables. In other words, more sequential solutions of ODEs are done by ode15s since the ODE integrator is called more by the optimization algorithm due to higher number of decision variables.

Table 6.1. Performance Index Values and the CPU times for the Solution of Problem 1 via the Direct CVO Method.

Ntgrid	**J***	**CPU (s)**	***Ntgrid***	**J***	**CPU (s)**
11	16.7740	732	**41**	16.7782	7130
21	16.7563	2226	**51**	16.7801	8137
31	16.7616	5301	**101**	16.8741	18157

The optimal state and control trajectories are given on Figures 6.2 - 6.7. As seen on these figures, the control profiles become fluctuant with increasing *Ntgrid*. In order to prevent or lessen those fluctuations a smoothing filter or a smoothing penalty function can be applied to the control vector. It should be kept in mind that the optimum decision variables found by the optimizer are shown at the grid points by small black dots and the control trajectories between the grid points are their spline interpolants.

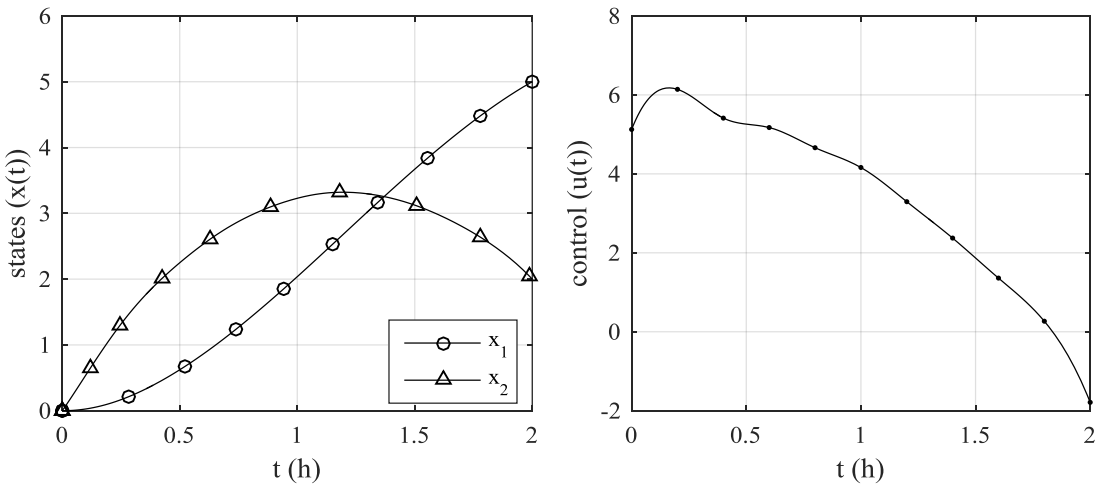


Figure 6.2. Optimal state and control trajectories of Problem 1 via CVO Method for *Ntgrid*=11.

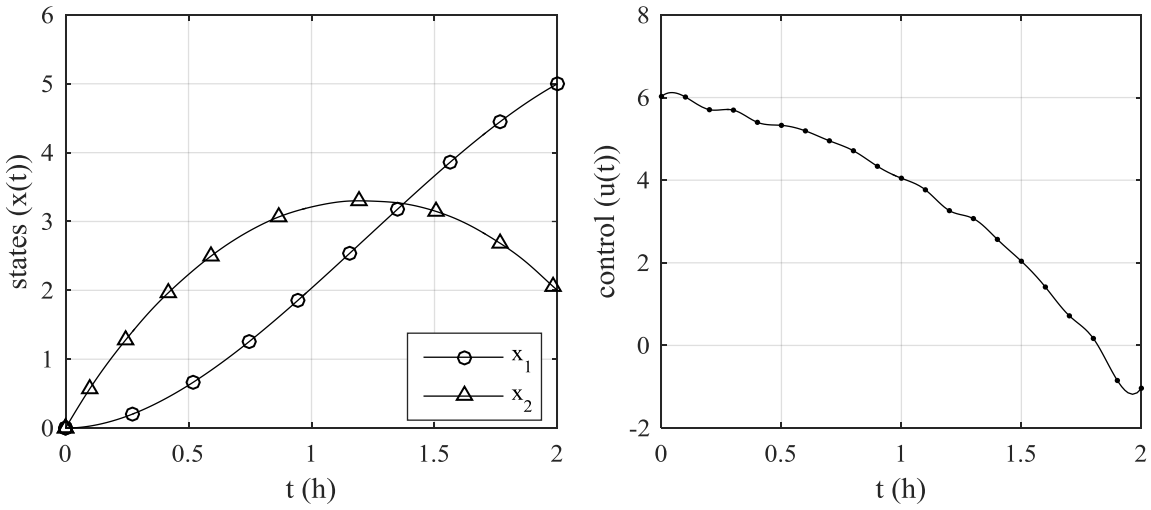


Figure 6.3. Optimal state and control trajectories of Problem 1 via CVO Method for *Ntgrid*=21.

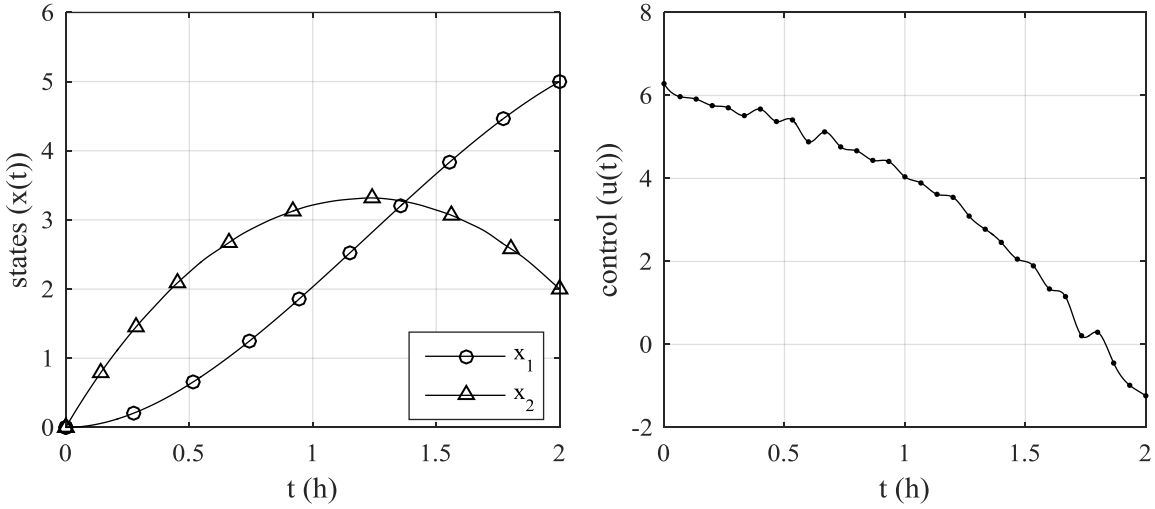


Figure 6.4. Optimal state and control trajectories of Problem 1 via CVO Method for *Ntgrid*=31.

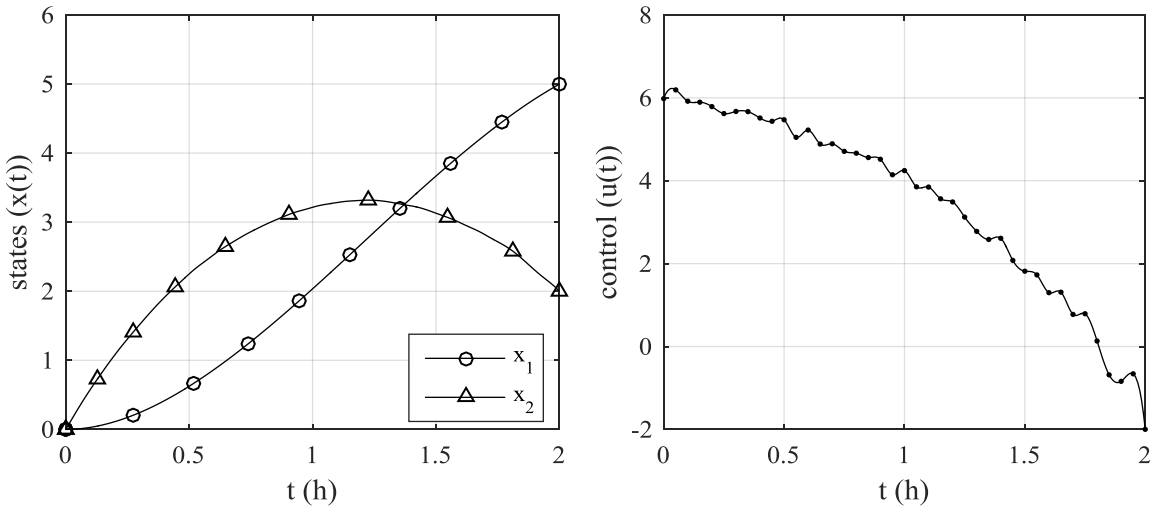


Figure 6.5. Optimal state and control trajectories of Problem 1 via CVO Method for *Ntgrid*=41.

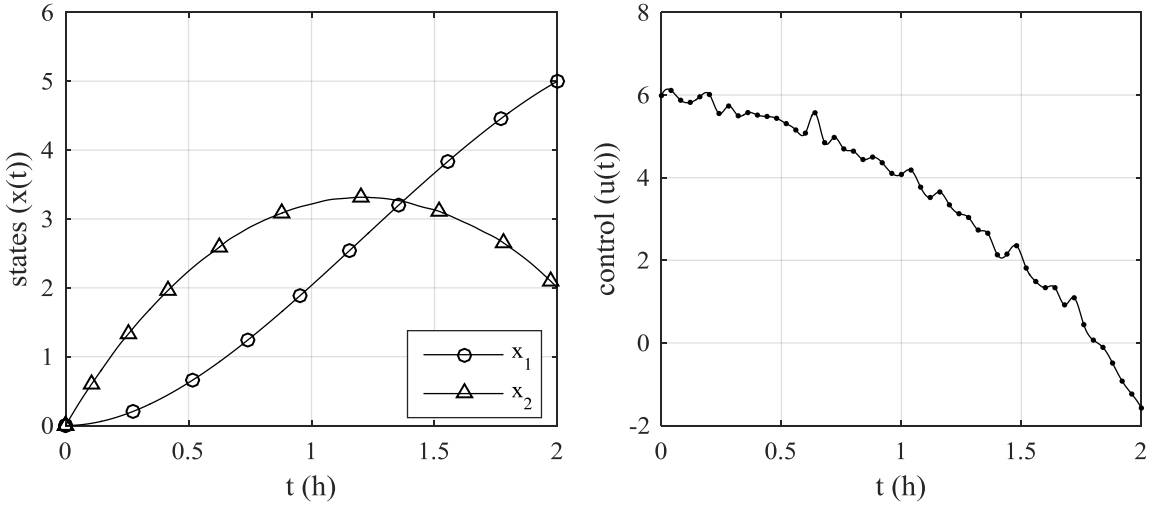


Figure 6.6. Optimal state and control trajectories of Problem 1 via CVO Method for *Ntgrid*=51.

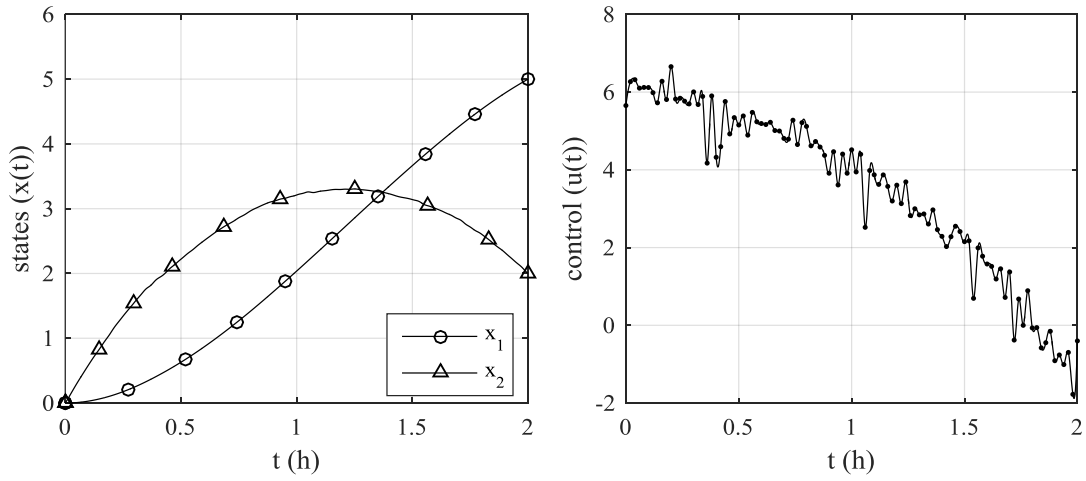


Figure 6.7. Optimal state and control trajectories of Problem 1 via CVO Method for *Ntgrid*=101.

To sum up, the CVO method is computationally expensive, has accuracy problems and can be trapped into local optimum with very fluctuant trajectories.

6.1.3. Solution of Problem 1 via the CVP Method

The problem was solved by the CVP Method by assuming different functional forms, e.g., guessed functional approximations of the yet unknown control trajectory, u(t), as discussed in Chapter 4.1. Initial guess of all parameters was taken as zero. The functional forms used, the optimal parameter values, and the corresponding performance indexes are given in Table 6.2.

Table 6.2. Functional forms used in the CVP to solve Problem 1 and the corresponding performance indexes.

Functional form	**u*(t)**	**J***
(i) Constant + exponential function	$-43.04 + 50.46\ e^{0.08\ t}$	17.0959
(ii) Constant + exponential term	$7.29 - 1.19\ e^{t}$	16.7508
(iii) Constant + 1st order polynomial + exponential term	$13.49 - 6.76\ t - 7.74\ e^{1.06\ t}$	16.7812
(iv) 2nd order polynomial	$5.91 - 0.10\ t - 1.73t^2$	16.7582
(v) 3rd order polynomial	$6.13 - 1.47\ t - 0.02t^2 - 0.57\ t^3$	16.7509
(vi) 4th order polynomial	$6.10 - 1.17\ t - 0.67\ t^2 - 0.07\ t^3 - 0.12\ t^4$	16.7507

Functional forms (i) and (ii) were expected to find the global optimum since they are in the same form as Equation (6.1) which is the analytical solution. However, the optimization sequence (““fmincon”” and DE) found a local minimum by using the functional form (i) due to its extra parameter in the exponential term. In order to satisfy the parameters of Equation (6.9), the optimizers have to find that parameter as equal to “1”, however this is not satisfied by the optimizers. Hence, in the functional form (ii), that parameter was fixed as “1” and the optimization sequence finds the two parameters, and hence the global optimum, correctly. Moreover, an additional parameter was used to the functional form (i) to make it (iii). In this case, a solution better than (i) was found but still it was not close to the global optimum. Optimum state and control trajectories of functional forms (i), (ii), and (iii) are given on Figures 6.8, 6.9, and 6.10.

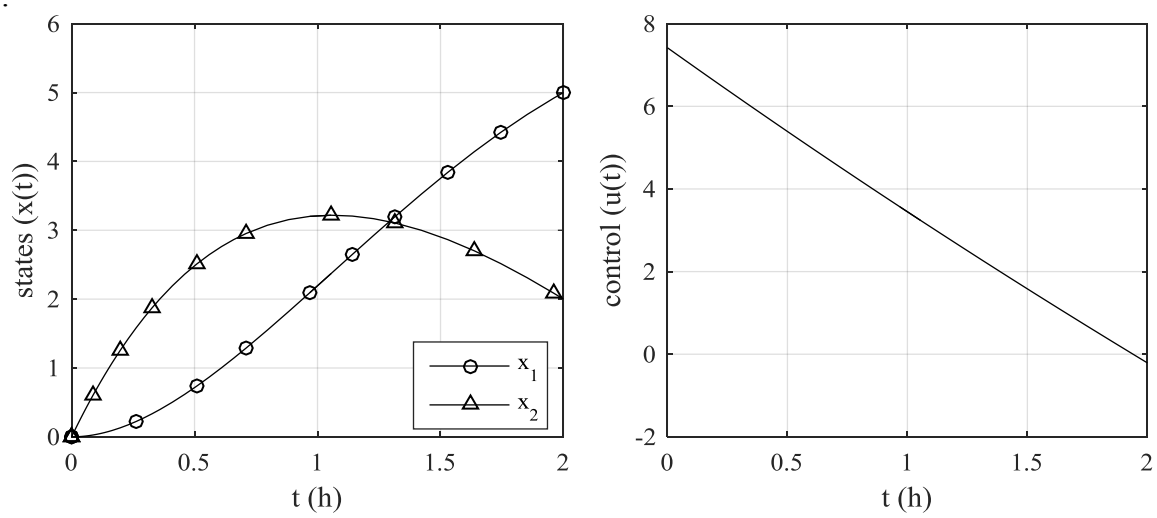


Figure 6.8. Optimal state and control trajectories of Problem 1 via the CVP Method by the functional form (i).

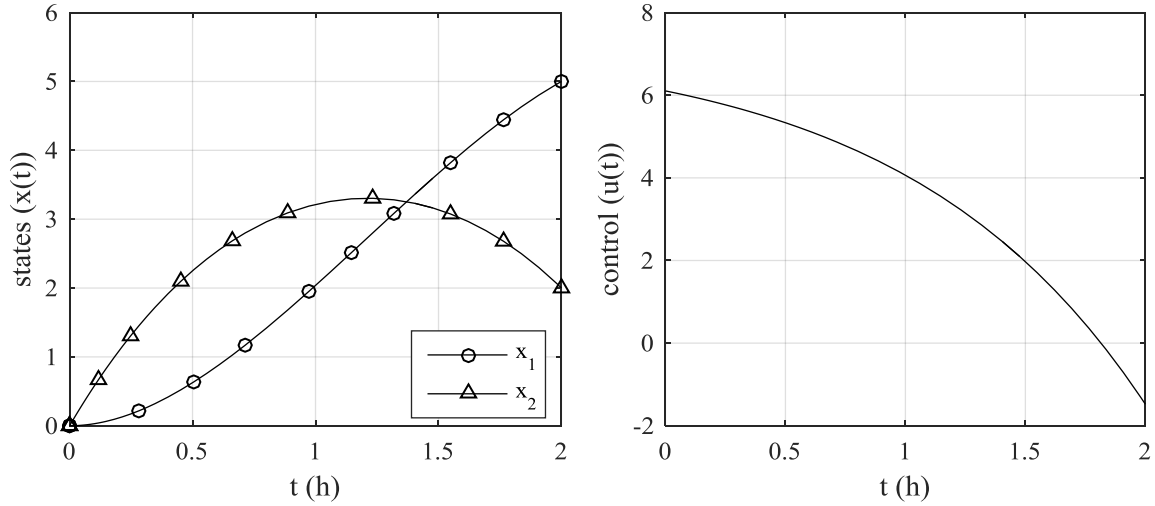


Figure 6.9. Optimal state and control trajectories of Problem 1 via the CVP Method by the functional form (ii).

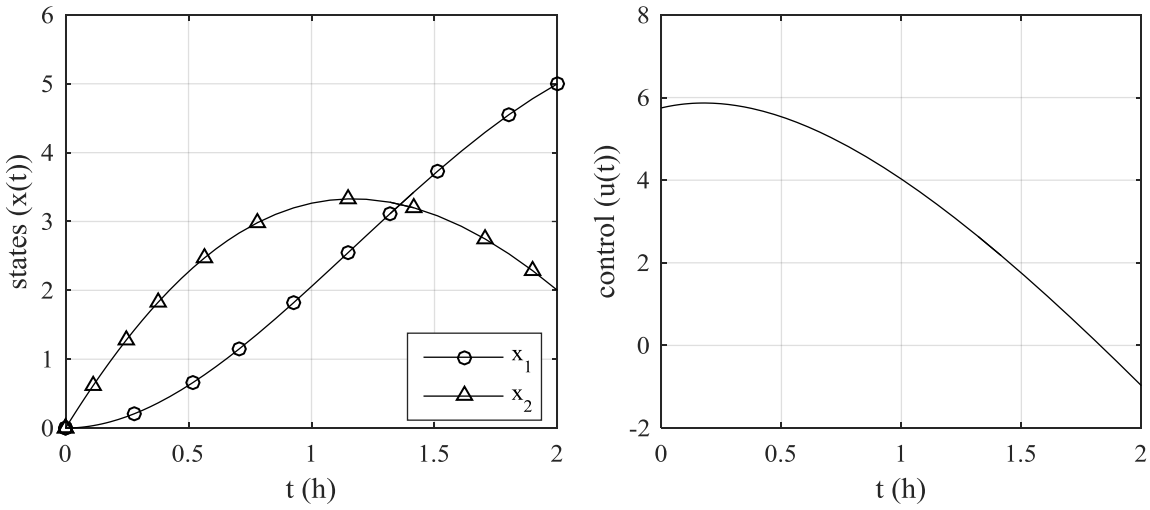


Figure 6.10. Optimal state and control trajectories of Problem 1 via the CVP Method by the functional form (iii).

In the functional forms of (iv), (v), and (vi), the optimum control trajectory is assumed to have 2nd, 3rd, and 4th order polynomial forms, respectively. Increasing the polynomial order yields better performance indexes as expected. The 3rd and 4th order polynomial functions (functional forms (v) and (vi)) give the same performance index up to three digits after the decimal, and (vi) has the J*=16.7507, equal to that of the analytical solution. The functional form (iv) is given on Figure 6.11, while (v) and (vi) are given on Figure 6.12, and they are impossible to discriminate.

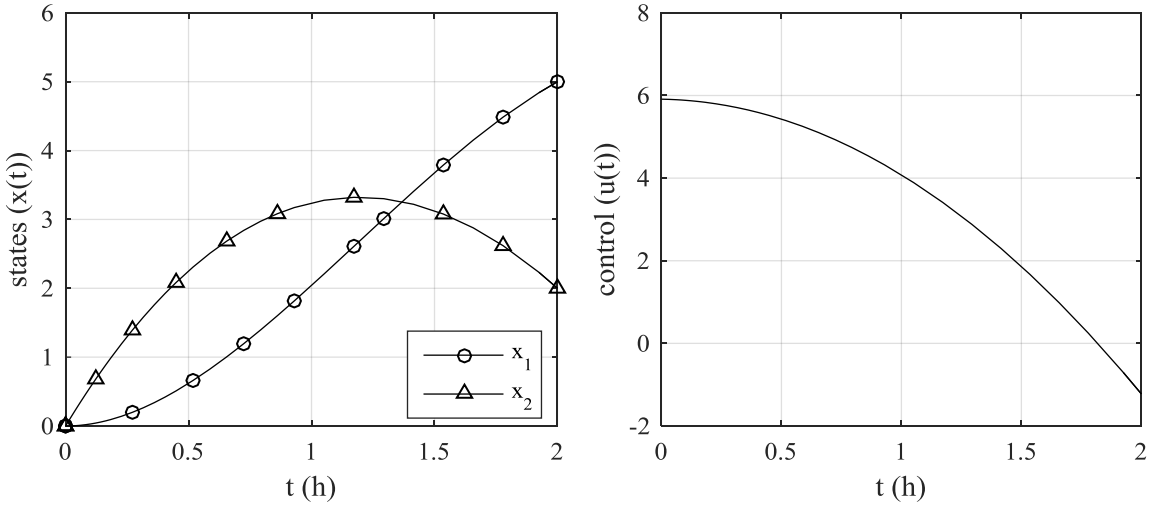


Figure 6.11. Optimal state and control trajectories of Problem 1 via the CVP Method by the functional form (iv).

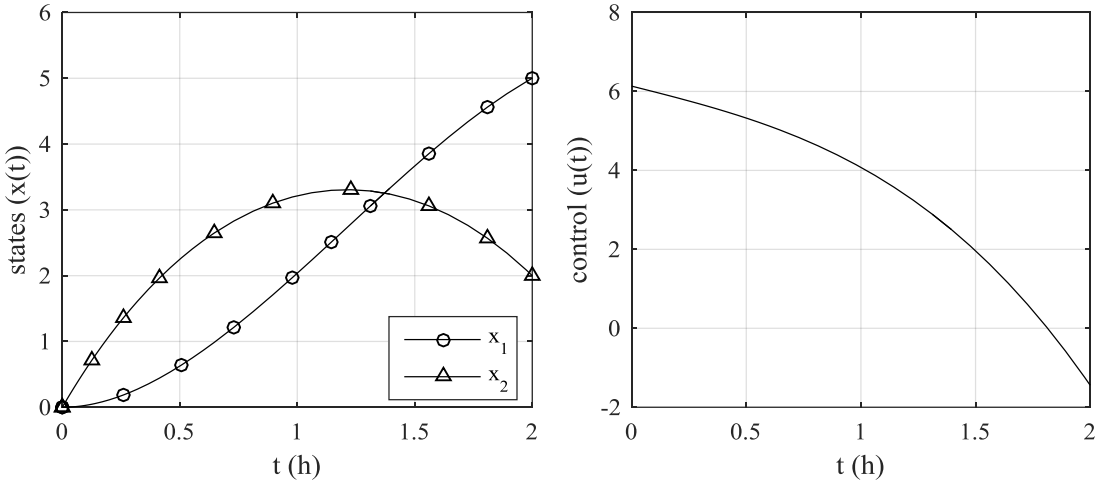


Figure 6.12. Optimal state and control trajectories of Problem 1 via the CVP Method by the functional forms of (v) and (vi).

To summarize, the CVP method can find the global optimum point if it has few parameters. However, its performance depends highly on the functional form assumed. When the shape of the control trajectory is known, the CVP works well. On the other hand, if the control trajectory does not have a known functional form or control vector is too complicated (too difficult to be expressed as a known functional form) then the CVP method is not reliable.

6.1.4. Solution of Problem 1 via the CVP-DCT Method

The problem was solved with *Ntgrid*=11 by taking different number of Discrete Cosine Transform Coefficients (DCTCs). Initial guess was selected as $u(t) = 0$. The upper and lower bounds on decision variables are given as -20 and $+20$ respectively. Note that these bounds belong to DCTCs which were expected to be within those limits. Furthermore, as explained before, the optimizer DE disregards the bounds after generation of the initial population. All calculations were done with the same equality-constraint penalty parameter (μ_e) and the final-state specifications Equation (6.4) are exactly satisfied. The value of μ_e was determined empirically and fixed at 10 for this problem. Optimum performance indexes are given in Table 6.3.

Table 6.3. Optimal performance indexes of Problem 1 by the CVP-DCT on 11 time grids for different *Ndctc*.

Ndctc	**J***	***Ndctc***	**J***	***Ndctc***	**J***
1	25.8930	5	16.7612	9	16.7558
2	17.1284	6	16.7545	10	16.7648
3	16.8493	7	16.7529	11	16.7632
4	16.7740	8	16.7516		

The J* values are compared with the analytical solution (J*=16.7507) and the relative differences were calculated. With only 2 DCTCs, the CVP-DCT method gives the performance index with a 2.2% relative difference and the relative difference percentage decreases to 0.5%, 0.13%, and 0.06% with 3, 4, and 5 coefficients, respectively.

According to the optimum values of performance indexes, up to the 8 DCTCs, the J* values decreased as *Ndctc* increases. Runs with *Ndctc*=9, 10, and 11 find extremum points relatively worse than *Ndctc*=8. This behavior was investigated in upcoming section. The optimal state and control trajectories obtained via the CVP-DCT method are shown in Figures 6.13-6.23.

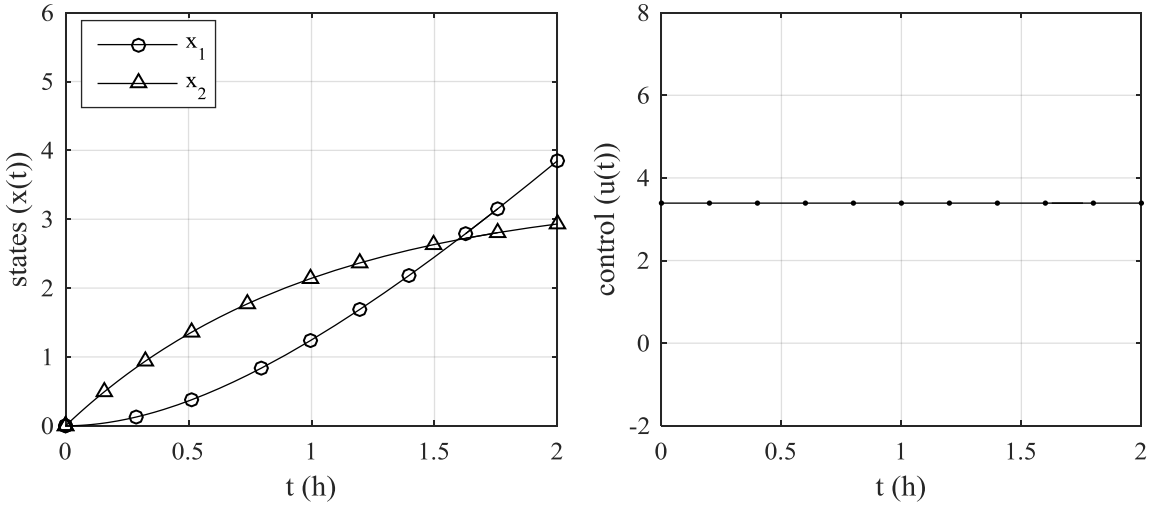


Figure 6.13 Optimal state and control trajectories of Problem 1 obtained with *Ndctc* =1 and *Ntgrid*=11.

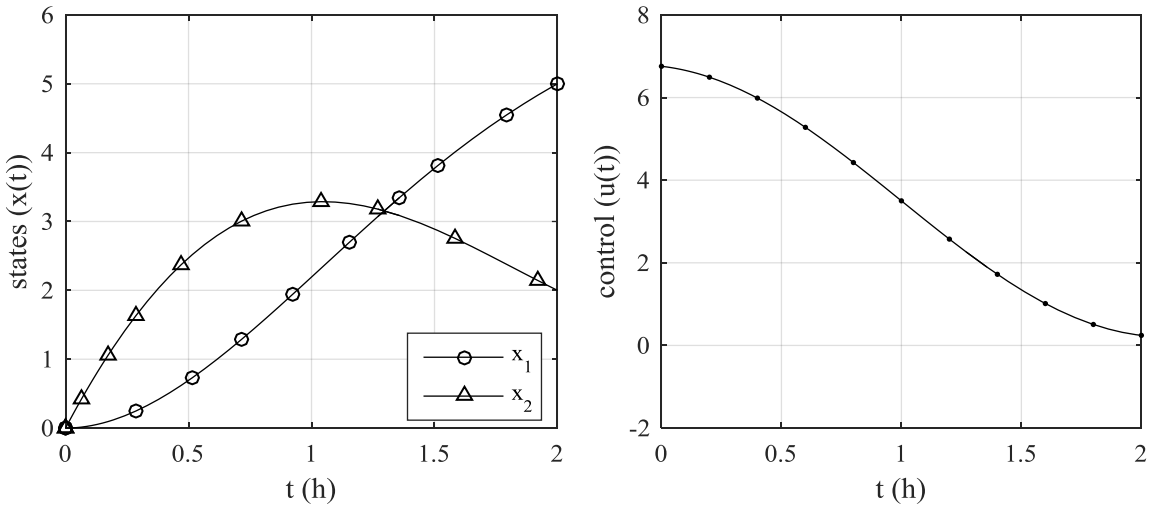


Figure 6.14 Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=2 and *Ntgrid*=11.

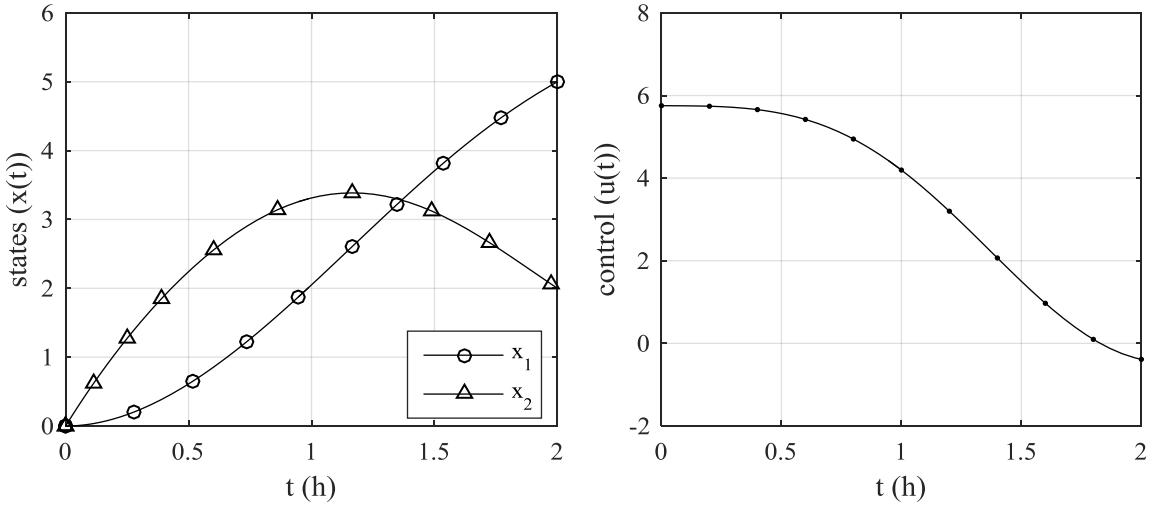


Figure 6.15 Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=3 and *Ntgrid*=11.

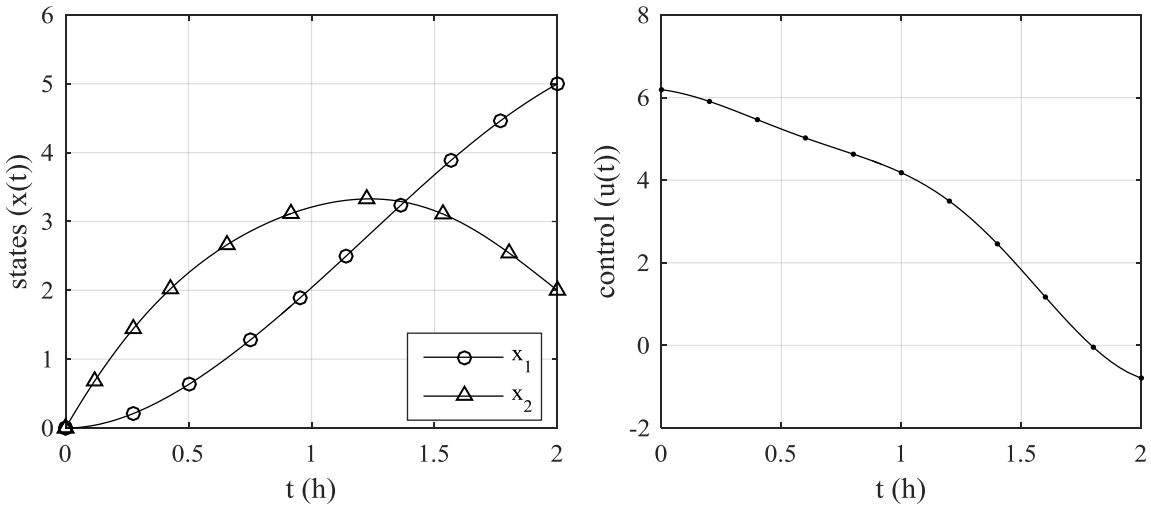


Figure 6.16. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=4 and *Ntgrid*=11..

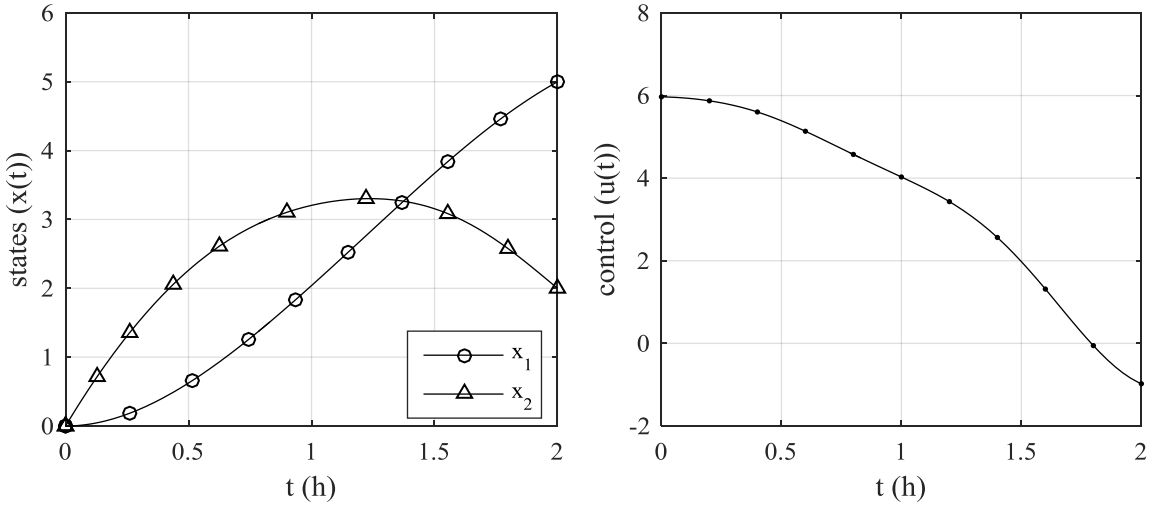


Figure 6.17. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=5 and *Ntgrid*=11.

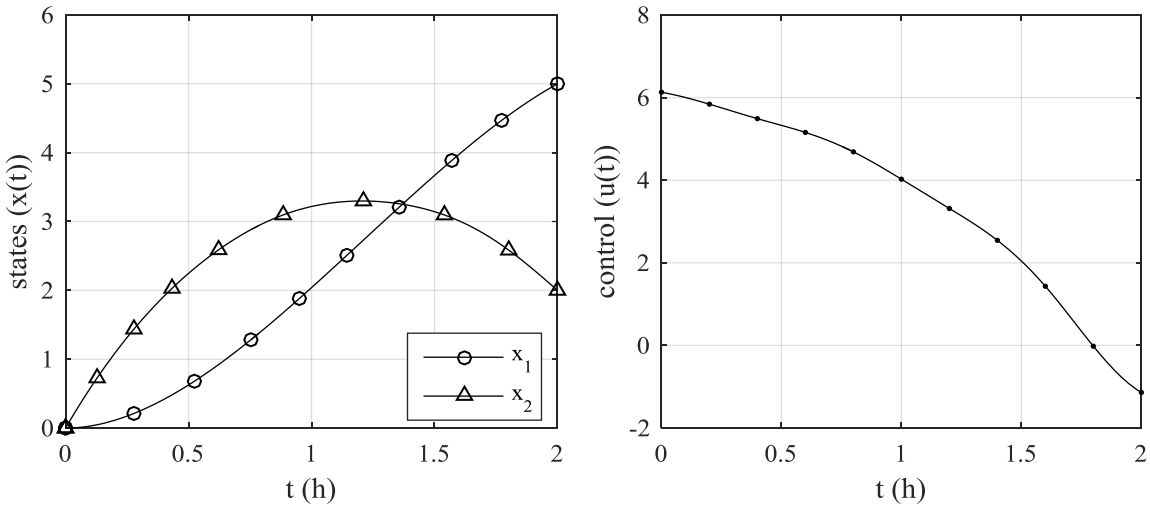


Figure 6.18. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=6 and *Ntgrid*=11.

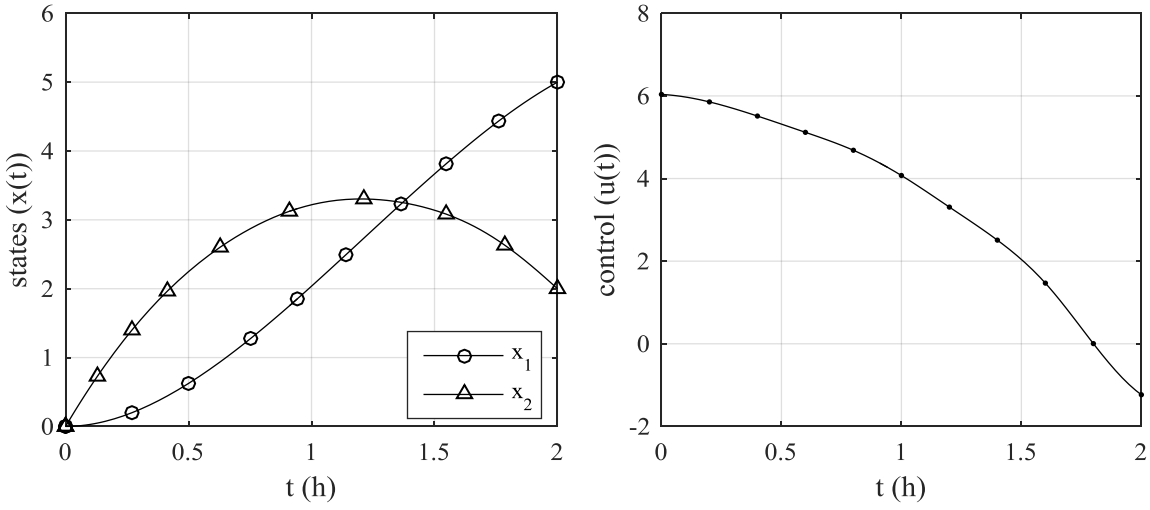


Figure 6.19. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=7 and *Ntgrid*=11.

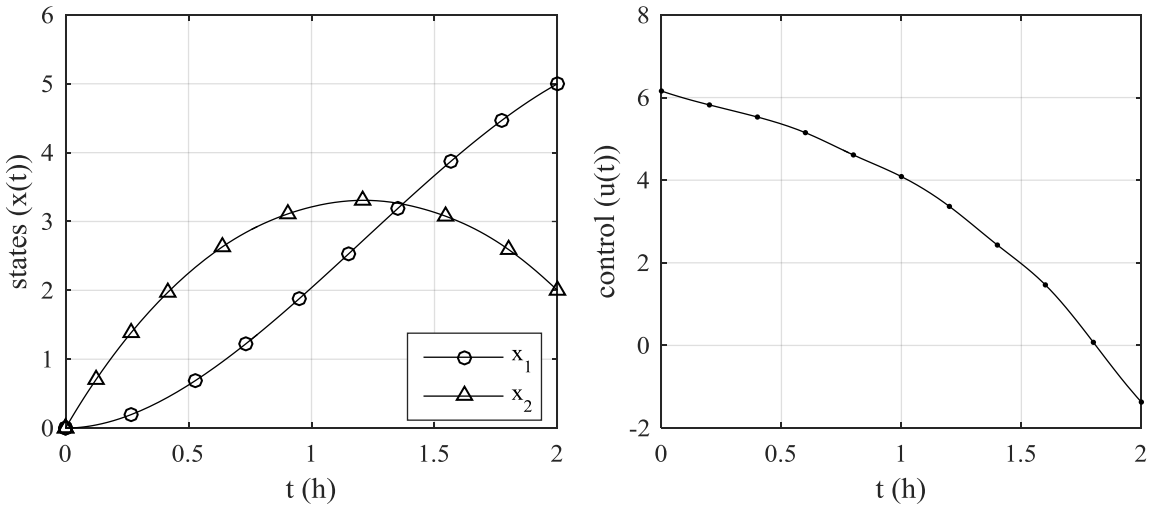


Figure 6.20. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=8 and *Ntgrid*=11.

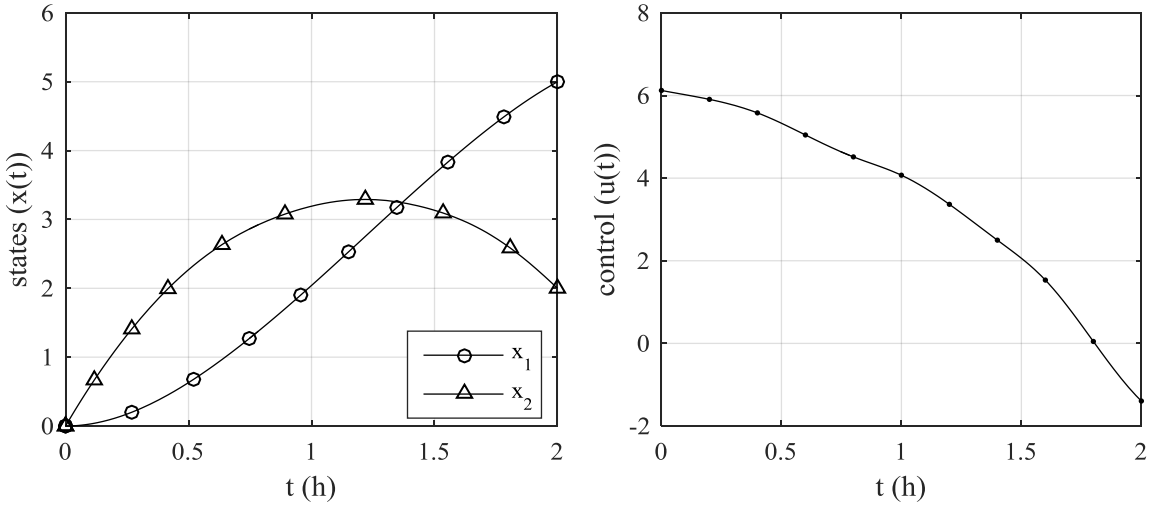


Figure 6.21. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=9 and *Ntgrid*=11.

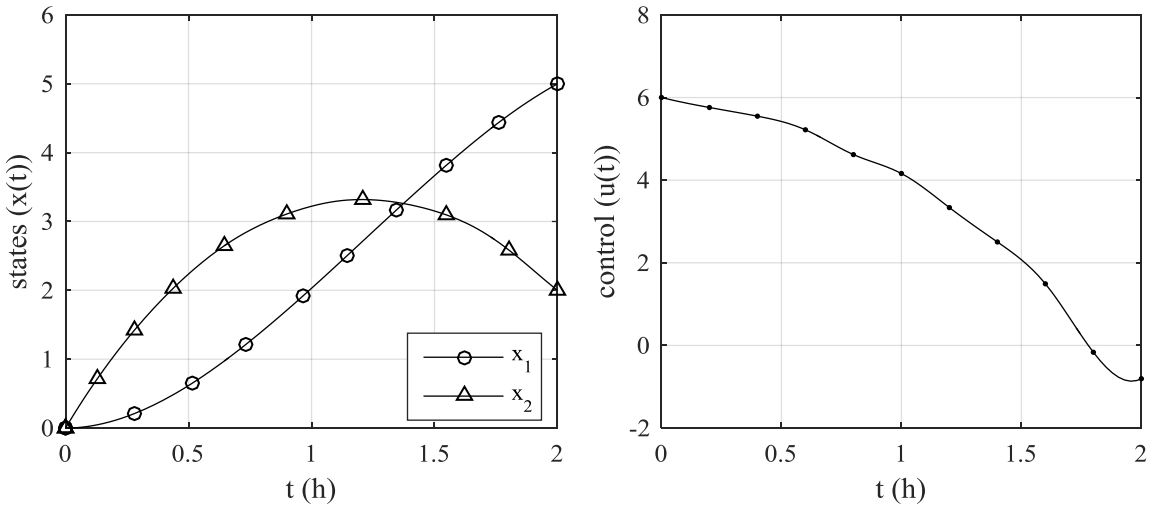


Figure 6.22. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=10 and *Ntgrid*=11.

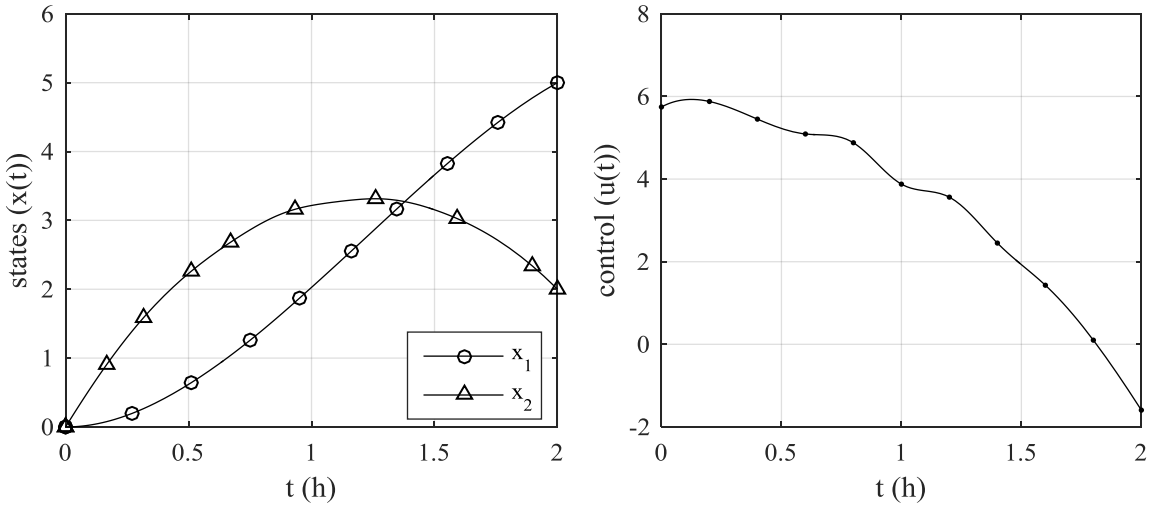


Figure 6.23. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=11 and *Ntgrid*=11.

As stated in Chapter 5, the main idea of using the CVP-DCT method is the compressive parameterization of the control vector, parsimoniously, with fewer parameters, by capitalizing on the information-compression property of the DCT. Control vector is decompressed with selected *Ndctc*. Optimizer deals only with *Ndctc* parameters. Then, the compressively-parameterized control vector is uncompressed, i.e., completed to *Ntgrid* by zero padding via Inverse DCT (i-DCT). The i-DCT unfolds the control vector which is compressed in the DCTCs and thus generates the control trajectory $\mathbf{u}_{\text{grid}}$. As described in

Chapter 2.1, using higher number of DCTCs achieves decompressing to more complex functional forms for the control vector. Therefore, it is expected to generate more complex forms of $\mathbf{u}_{grid}$ by using larger *Ndctc* values. In other words, using more Ndctc needs less zero padding. Let us have a look at optimum DCTCs when *Ndctc*=2, 5, and 11 runs given in Table 6.4. Results of those runs were already represented in Figures 6.14, 6.17, and 6.23. As shown in Table 6.4, values of the 1st and the 2nd coefficients are inversely proportional to *Ndctc*. As *Ndctc* increases, the sensitivity of the performance index to DCTCs decreases. Inclusion of DCTCs further from the first few coefficients caused reconstructed trajectories to be more and more nonsmooth. This fact was combined with the fact that as the number of decision variables increases, optimizing the objective functional becomes more difficult (more iterations, less probability of obtaining the global minimum), and thus it leads to unsmooth control profiles.

Table 6.4. Optimal DCTCs of the control vector of Problem 1 with *Ndctc*=-2, 5, and 11.

Ndctc	2	5	8
1	11.61	11.29	10.08
2	7.72	7.34	5.22
3	0	-1.87	-3.07
4	0	0.96	-0.90
5	0	-0.41	-1.92
6	0	0	-0.51
7	0	0	-1.74
8	0	0	-1.38
9	0	0	-0.91
10	0	0	-2.07
11	0	0	-0.91

Therefore, to smooth the control trajectory, a penalty function was applied. The Smoothing Penalty Function (SP) is given by Equation (6.9), where μ_s represents the smoothing penalty parameter and Δ^2 is the second-order differencing operator.

$$\mathrm{SP} = \mu_s \left\| \Delta^2 \mathbf{u}_{\mathrm{grid}} \right\| = \mu_s \sqrt{\sum_{k=2}^{N-1} (u_{k+1} + 2\,u_k + u_{k-1})^2} \tag{6.10}$$

The augmented performance index given in Equation (6.8) now becomes as follows by addition of the smoothing penalty function Equation (6.10)

$$J = x_3(t_f) + \mathrm{ECP} + \mathrm{SP} \tag{6.11}$$

Runs with 9, 10, and 11 DCTCs are repeated by adding smoothing function with μ_s value taken as 1 for each run (this value is determined empirically, just based on of visual smoothness of the control trajectory). The calculated performance indexes with SP are compared with the earlier runs without smoothing in Table 6.5.

Table 6.5. The effect of SP on the optimal performance indexes of Problem 1 obtained on 11 time grids for different *Ndctc* values.

Ndctc	**J* without SP (from Table 6.3)**	**J* with SP**
9	16.7558	16.7962
10	16.7648	16.7851
11	16.7632	16.7955

The optimum state and control trajectories are represented on Figure 6.24, 6.25 and 6.26. On one hand, the use of SP made the optimal control trajectories smoother (e.g. compare Figures 6.21. and 6.24) on the other hand, the optimal performance indexes got larger than those calculated without the SP. Whether this smoothing function was required or not was investigated in the following section.

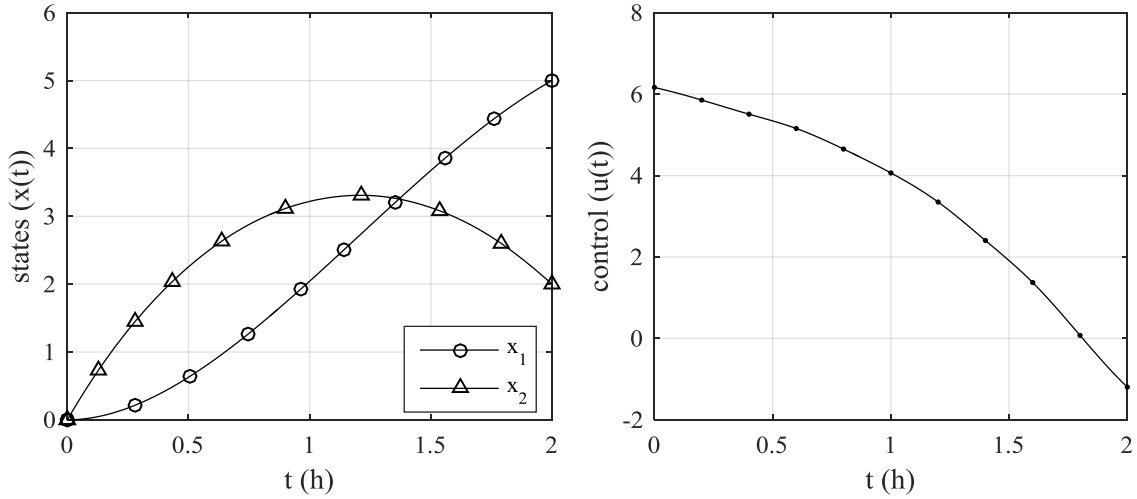


Figure 6.24. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=9 and *Ntgrid*=11 with SP.

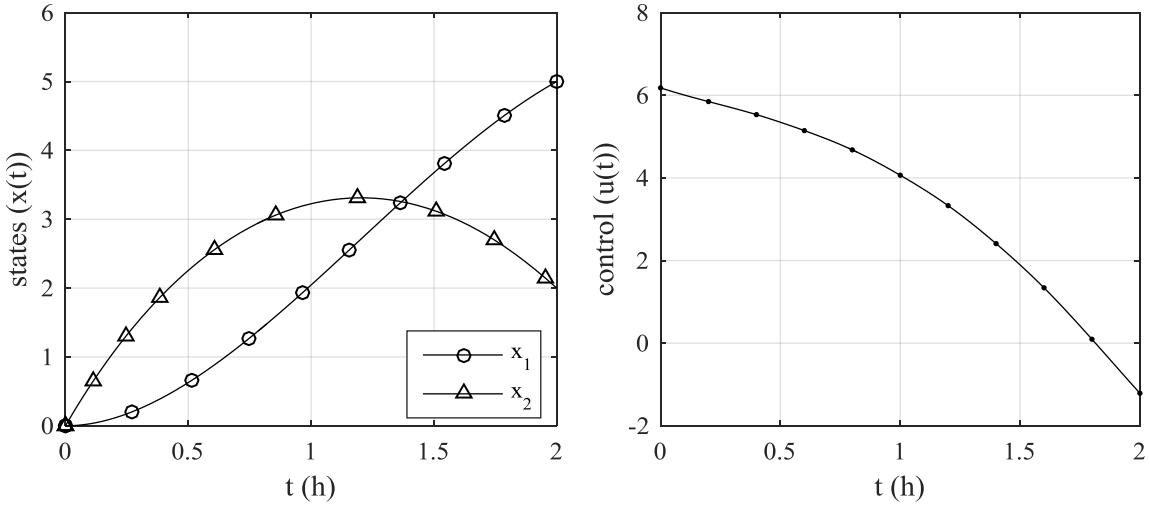


Figure 6.25. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=10 and *Ntgrid* =11 with SP.

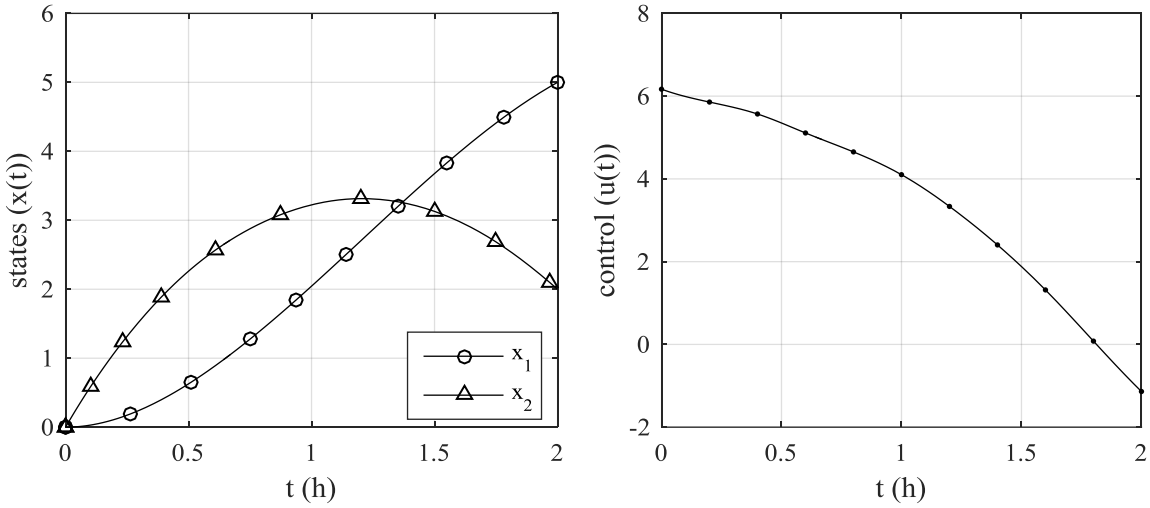


Figure 6.26. Optimal state and control trajectories of Problem 1 obtained with *Ndctc*=11 and *Ntgrid* =11 with SP.

6.1.4.1. Verification of the use of SP for the CVP- DCT. In this section, whether the smoothing function was indeed required or not was investigated by introducing the analytical solution of Problem 1 as the initial guess to the CVP-DCT method. As shown in the previous sections, the CVP-DCT method yielded worser performance indexes after the 8th DCTC because using more number of coefficients create a fluctuant control trajectory for Problem 1. Applied smoothing function made the trajectory smoother but somewhat deteriorated the performance index. The idea of this test was this: The optimum control vector obtained analytically has a smooth profile so that the major DCTCs, which appear also early in the DCTC series, of this analytical control trajectory must be sufficient (or very close to be sufficient) to make the optimization algorithm converge near the global optimum. Initial guess was supplied by evaluating the analytical solution, u*(t), given in Equation (6.1) on each grid point to obtain the optimum control vector, $\mathbf{u}^*_{\text{grid}}$. Then DCT of $\mathbf{u}^*_{\text{grid}}$ was taken and Ndctc parameters were selected at each run and vectors were zero padded to be used as initial guess. The optimum performance indexes calculated by this approach are given in Table 6.6. According to results, without the SP, the CVP-DCT may find better performance indexes if a precise initial guess is supplied as full profile, $\mathbf{u}_{\text{grid}}$, to the optimizer. The results obtained by using initial guess of $\mathbf{u}_{\text{grid}} = \mathbf{0}$ (Table 6.3) and the results obtained by taking DCTC of $\mathbf{u}^*_{\text{grid}}$ (Table 6.6) can be compared. By this comparison, the test shows that up to the 8th DCTC, the J* values and the control profiles are the same and this

demonstrates that the optimizer can indeed converge to the optimum parameters that correspond to the global optimum, considering the chosen termination tolerance limits).

Table 6.6. Performance indexes of Problem 1 obtained by using the analytical solution as initial guess of the CVP-DCT on 11 time grids.

Ndctc	**J***	***Ndctc***	**J***	***Ndctc***	**J***	***Ndctc***	**J***
1	25.8930	**4**	16.7740	**7**	16.7528	**10**	16.7509
2	17.1284	**5**	16.7612	**8**	16.7516	**11**	16.7507
3	16.8493	**6**	16.7544	**9**	16.7513		

Apparently, the CVP-DCT method did not definitely require a smoothing penalty especially with lower number of DCT coefficients. Moreover, runs by 9, 10, and 11 DCTCs yielded both better performance indexes and smoother optimal control profiles. Therefore, Table 6.6 proves that, supplying a very good initial guess yields much better performance indexes and the attainment of global solution is indeed possible. However, for larger *Ndctc* values, the DE algorithm, unfortunately, may be unable to find the global solution, especially if the supplied initial control profile, $\mathbf{u}_{\text{grid}}$ is far from the true global solution.

6.1.4.2 The Relationship Between *Ntgrid* and *Ndctc*. In this section, the relationship between Ntgrid and Ndctc is briefly analyzed. Problem 1 was solved with six different Ntgrid for five different Ndctc values. The corresponding optimum performance indexes for each Ndctc and Ntgrid pair are given on Table 6.7. According to the J* values, with Ndctc=2, 4, 6, and 8 as Ntgrid increased J* values increased as well. Therefore, CVP-DCT method works better with Ntgrid=11 for this problem. With Ndctc=10, fluctuant trajectory led to a worse J* when Ntgrid=11 and this case is discussed in section 6.1.3.

Table 6.7. Performance indexes of Problem 1 obtained with different *Ndctc* and *Ntgrid* values.

Ndctc	***Ntgrid***					
	11	21	31	41	51	101
2	17.1284	17.1643	17.1489	17.1520	17.1542	17.1581
4	16.7740	16.8149	16.7850	16.7868	16.7879	16.7902
6	16.7545	16.7932	16.7586	16.7594	16.7599	16.7610
8	16.7516	16.7527	16.7533	16.7538	16.7541	16.7547
10	16.7648	16.7519	16.7526	16.7524	16.7525	16.7528

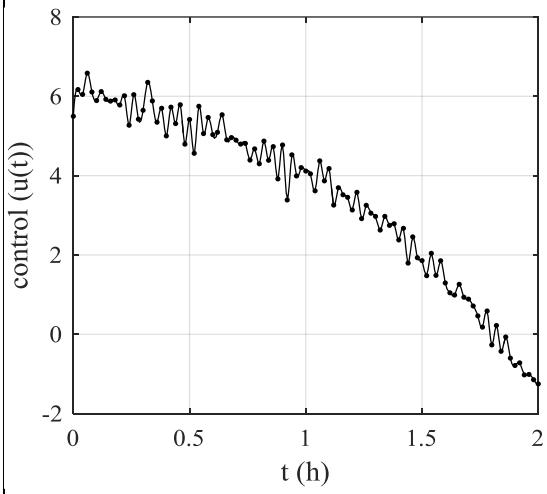
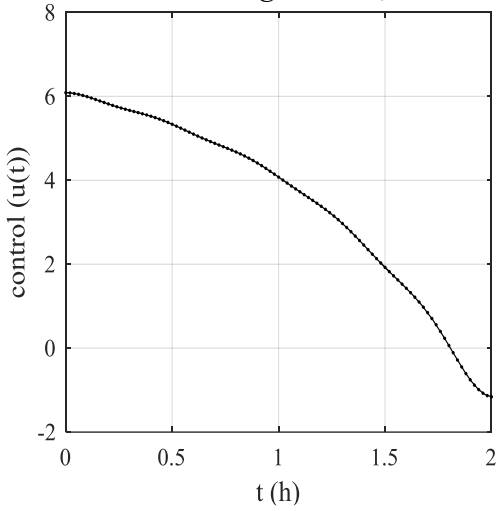
The CPU time was directly affected by *Ndctc* (which is the number of optimization decision variables), however, the change in the CPU time with *Ntgrid* for constant *Ndctc* was negligible. Using more *Ndctc* requires more CPU time due to increased number of decision variables, which, in turn, increased the optimization iterations, and hence, the number of times the system ODEs are solved. The results given on Table 6.7 are as expected. Since, during the i-DCT operation, more and more elements are zero-padded with increasing *Ntgrid* for a fixed *Ndctc* value. More and more zero-padding led to more and more filtered (smooth but restricted) function approximation, which, in turn, yields less agile parameterization of the control vector.

6.1.5. A Further Illustration of the Power of the CVP-DCT Method in Compressive Parameterization of the Control Trajectory for the Problem 1

In this final section, we numerically compare the solutions obtained by CVO with *Ntgrid*=101 and CVP-DCT with *Ndctc*= 10 and *Ntgrid*= 101. This comparison fortifies that the CVP-DCT method has several advantages due to DCT's data compression power. Firstly, on the same time grids (*Ntgrid* = 101), CVP-DCT yielded a better performance index. Moreover, it used fewer (one tenth of the CVO) optimization decision variables. Comparison of the optimization statistics of both methods showed that ""fmincon"" required more function evaluations for CVO and this is mostly caused by the large dimension of the Hessian matrix of the objective function, which is 101×101. This means, in CVO, the optimizer has

to compute 101×101=10201 elements of the Hessian. matrix using finite differencing, each one of which requires the solution of the state ODE set. Furthermore, the DE algorithm could not find a solution within the desired tolerance limits because DE algorithm reached the maximum iteration and maximum functional evaluation limits set. With the CVO method, the ODE integrator was called 35 times more than it was called in the CVP-DCT method. Finally, the required CPU time was 16.5 times longer in the CVO method. The comparison of computational statistics along with the corresponding control trajectories are given in Table. 6.8.

Table 6.8. Comparison via Computational Statistics of the CVO and CVP-DCT for Problem 1.

CVO with *Ntgrid*=101	CVP-DCT with *Ntgrid*=101, *Ndctc*=10
	
Number of decision variables : 101	Number of decision variables : 10
J* = 16.8072	J* = 16.7528
““fmincon”” stats 54 iterations 11412 objective function evaluations	““fmincon”” stats 21 iterations 4421 objective function evaluations
DE stats 2500 iterations 50000 objective function evaluations	DE stats 686 iterations 13720 objective function evaluations
Number of ODE function evaluations: 36,731,678	Number of ODE function evaluations: 1,668,165
CPU: 5255 seconds	CPU: 317 seconds

6.2. The Catalyst Mixing Problem

This problem is proposed by (Gunn and Thomas, 1965) and determines the optimal mixing policy of two catalysts. The reaction, $S_1 \leftrightarrow S_2 \rightarrow S_3$, takes place in a plug flow reactor. The problem is formulated as follows. The control variable, u, represents the mixing ratio of two catalysts.

$$\dot{x}_1 = u\,(10x_2 - x_1) \tag{6.12}$$

$$\dot{x}_2 = u(x_1 - 10\,x_2) - (1 - u\,x_2) \tag{6.13}$$

Initial conditions are given as follows while final time conditions are set free.

$$\mathbf{x}(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.14}$$

The performance index is to maximize the production of the species S_3 at the final time (t_f) that is equal to 1 hour. Therefore, the performance index was defined as the minimization of the negative of the objective functional. In the following sections, the absolute values of the objective functional were given and thus higher values represents better performance indexes. The performance index to minimize is:

$$J = -1 + x_1(t_f) + x_2(t_f) \tag{6.15}$$

Since control variable, u(t), is defined as the mixing ratio and thus, the upper and lower bounds of u(t) are defined as 0 and 1.

$$0 \leq u(t) \leq 1 \tag{6.16}$$

For the solution via the CVO method, the upper and lower bounds were introduced to "fmincon". However, the DE algorithm disregards the upper and lower bounds as stated at the beginning of this chapter. Since in the CVP-DCT method, the decision variables are DCTCs, the upper and lower bound were set as -5 and +5 empirically for "fmincon". Therefore, to prevent a possible violation of the control variable bounds, a penalty function called Upper-Lower Penalty (ULP) was augmented to performance index. The minimum and maximum elements of $\mathbf{u}_{\text{grid}}$ are defined as $u_{\min}$ and $u_{\max}$. The ULP given in Equation

(4.8) was applied as given in Equation.(6.17) and μ_{ul} represents the penalty parameter which was taken as 10 in this problem.

$$\mathrm{ULP} = \mu_{ul}\big((\max(0, -u_{min}) + (\max(0, u_{max} - 1)\big) \tag{6.17}$$

The augmented performance index became,

$$J = -1 + x_1(t_f) + x_2(t_f) + \mathrm{ULP} \tag{6.18}$$

6.2.1. Solution of the Catalyst Mixing Problem via the CVO Method

The Catalyst Mixing problem was solved via the CVO method on 11, 21 and 31 time grids. Initial guess is selected as $u(t_k) = 0$, k=1, ..., *Ntgrid*. Even though the control variables on the time-grid nodes satisfied the upper-lower bounds given in Equation (6.17), the cubic-spline interpolation between the time nodes causes so-called violations. It should be noted that the aforementioned so-called violations of the control trajectory bounds were solely due to the adaptation of the cubic-spline option for the interpolation task. The performance index obtained by *Ntgrid*=11 is equal to 0.048488. The state and control trajectories with *Ntgrid*=11 are given below on Figure 6.27.

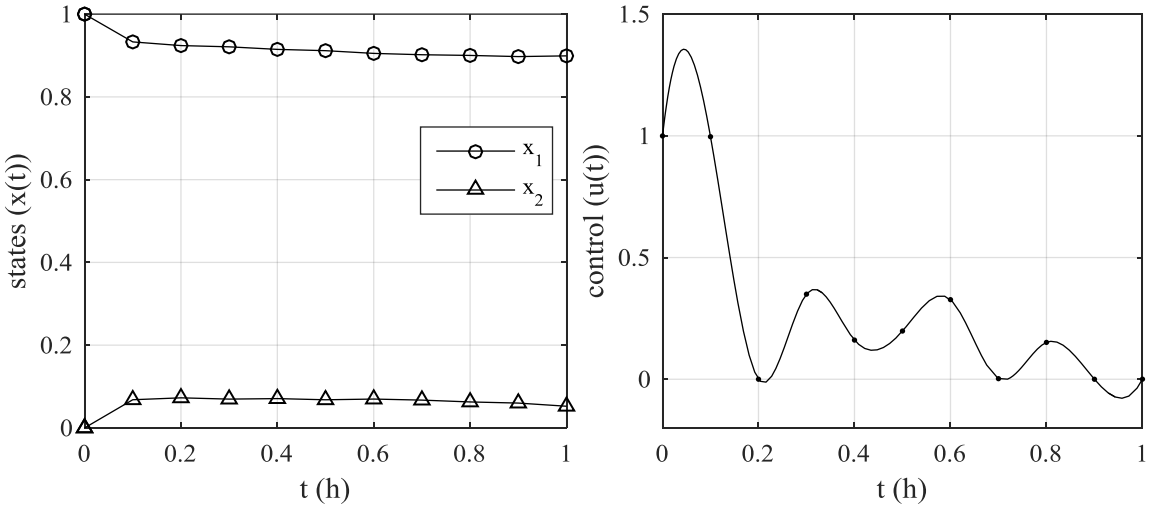


Figure 6.27. Optimal state and control trajectories of the Catalyst Mixing Problem via CVO for *Ntgrid* =11 with cubic-spline interpolation.

The same bound violations were observed until *Ntgrid*=51 which required much more CPU time. Therefore, the interpolation method was switched from the spline to pchip. With pchip interpolation, none of these so-called violations were observed since the optimization was based only on the grid points. Of course, the selected interpolation function had some effect on the solution of the ODEs and hence, in turn, on the objective functional value. Using the pchip interpolation option, the performance indexes are given in Table 6.9 and the corresponding optimum state and control trajectories are given on Figures 6.28 - 6.30.

Table 6.9. Performance Index Values for the Solution of the Catalyst Mixing Problem via the Direct CVO Method with Interpolation via spline.

Ntgrid	$\mathbf{J^*}$
11	0.048052
21	0.048063
31	0.048026

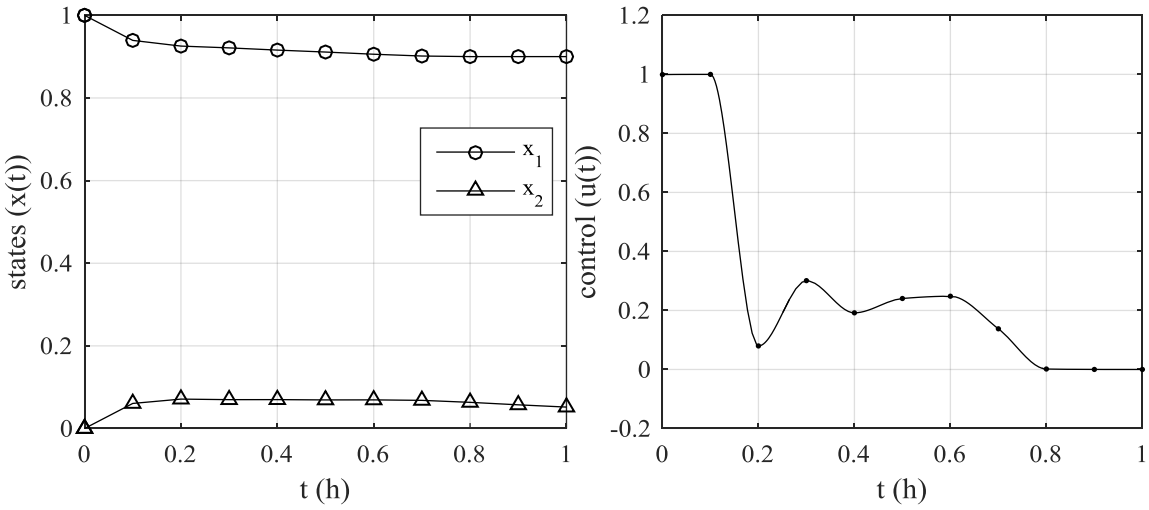


Figure 6.28. Optimal state and control trajectories of the Catalyst Mixing Problem via CVO for *Ntgrid*=11.

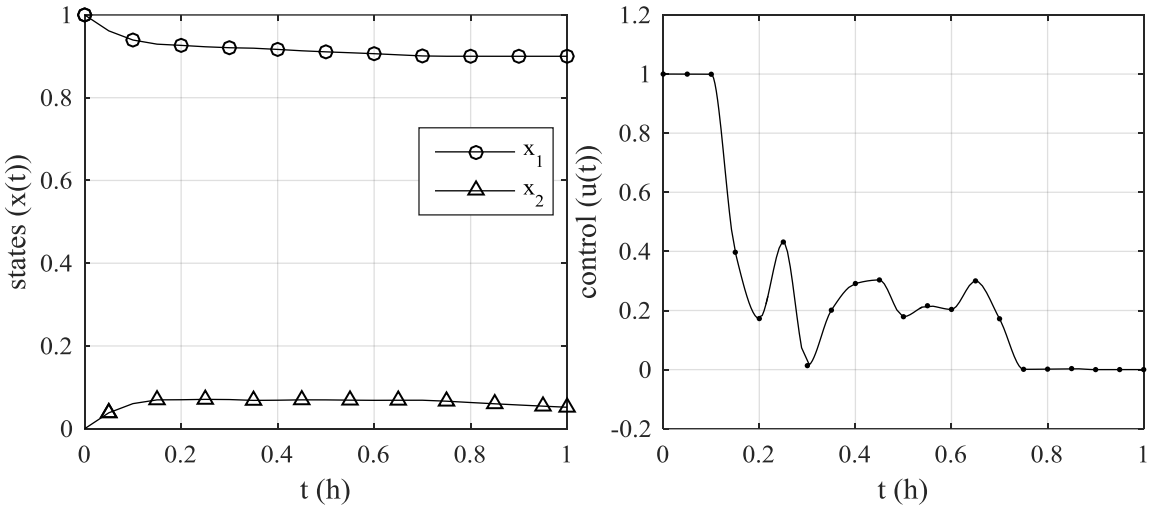


Figure 6.29. Optimal state and control trajectories of the Catalyst Mixing Problem via CVO for *Ntgrid*=21.

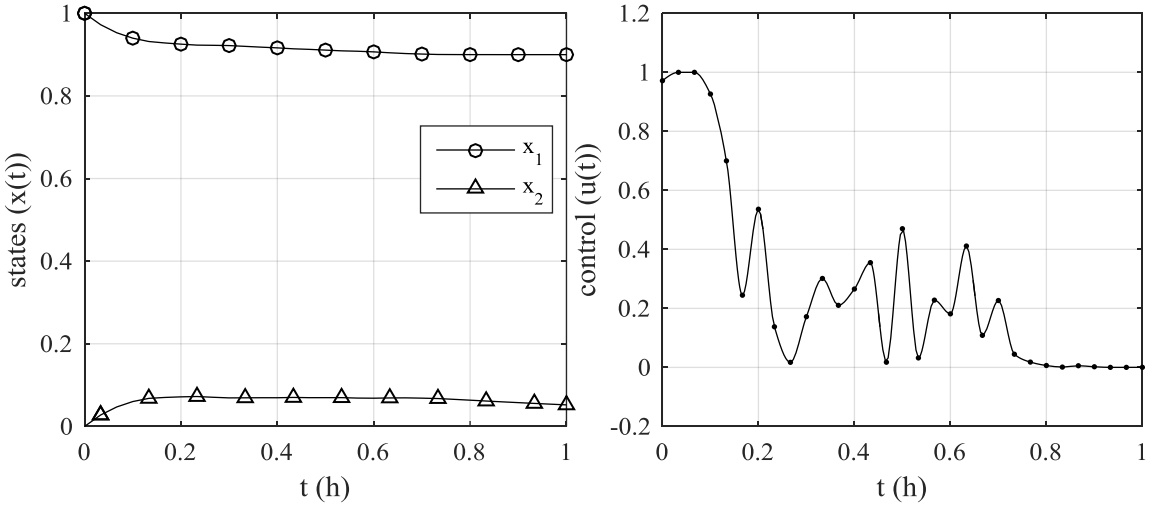


Figure 6.30. Optimal state and control trajectories of the Catalyst Mixing Problem via CVO for *Ntgrid*=31.

The control trajectories implied the possible presence of discontinuities and therefore fluctuant control profiles emerges. Hence, to smooth the control profile Smoothing Penalty Function, (SP), Equation (6.10) was augmented to performance index with a penalty parameter, μ_s, of 0.001. Performance indexes are given and compared in Table 6.10. Minimized performance indexes with the SP are worse than the ones without SP.

Table 6.10. Performance Index Values for the Solution of Catalyst Mixing Problem via the Direct CVO Method.

Ntgrid	**J^* without SP**	**J^* with SP**
11	0.048052	0.047544
21	0.048063	0.047756
31	0.048026	0.047971

The optimum state and control trajectories with SP were given on Figures 6.31 - 6.33. The SP made the performance index worse but the trajectories become smoother.

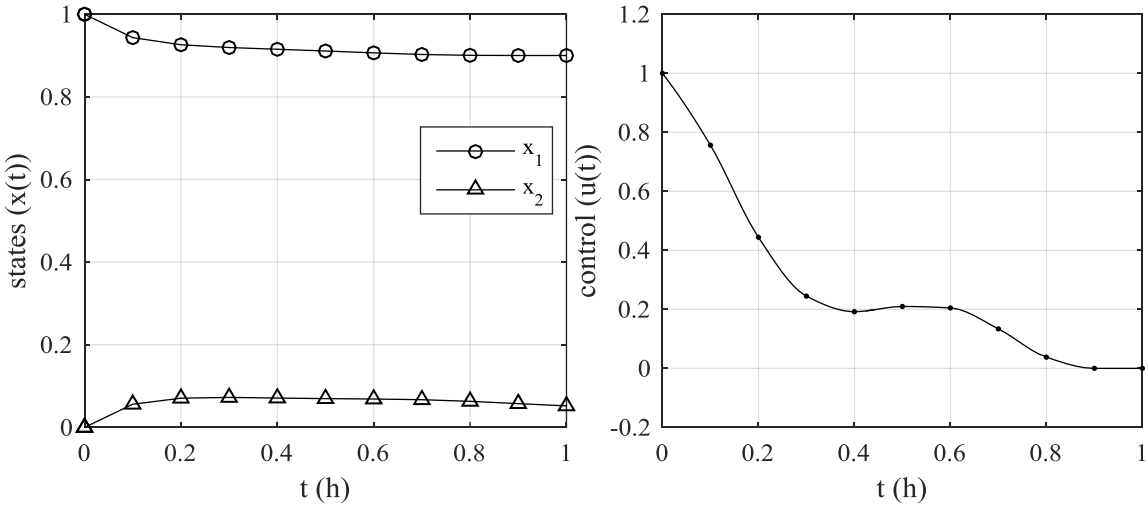


Figure 6.31. Optimal state and control trajectories of the Catalyst Mixing Problem via CVO for *Ntgrid* =11 with SP.

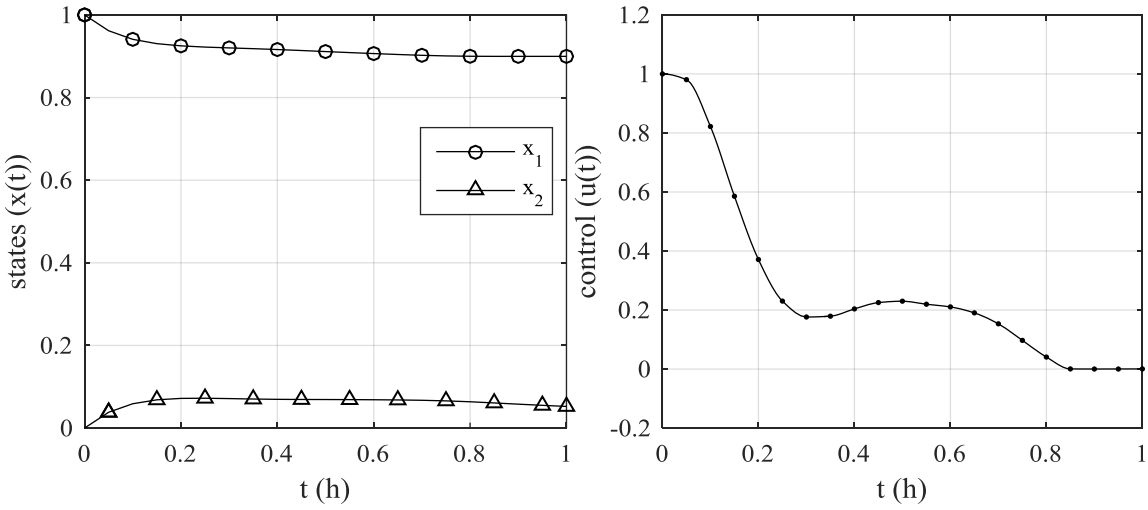


Figure 6.32. Optimal state and control trajectories of the Catalyst Mixing Problem via CVO for *Ntgrid* =21 with SP.

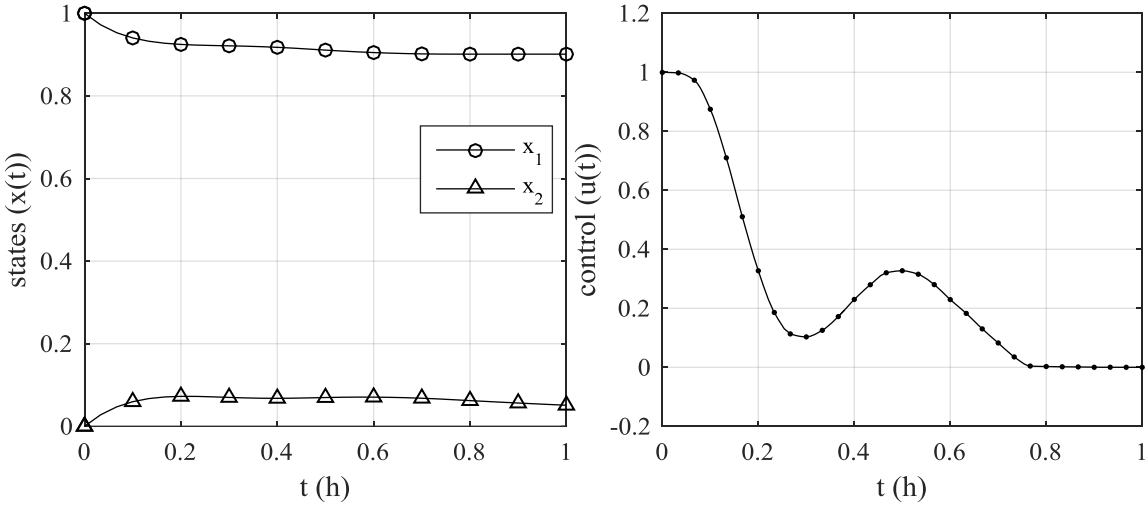


Figure 6.33. Optimal state and control trajectories of the Catalyst Mixing Problem via CVO for *Ntgrid* =31 with SP.

The control trajectories of the solutions above do not have a simple shape of a well-known function or a polynomial as opposed to the control trajectory of "Problem 1". In other words, oscillations and discontinuities makes the parameterization of the control trajectory much more difficult. Therefore, the CVP method with low number of parameters does not seem to be possible for this problem.

6.2.2. Solution of the Catalyst Mixing Problem via the CVP-DCT Method

The problem is solved with *Ntgrid* = 11 by taking different number of DCTCs. Initial guess is selected as $\mathbf{u}_{\text{grid}} = \mathbf{0}$. The optimum performance indexes obtained are given in Table 6.11. The control variable bounds are satisfied.

Table 6.11 The optimal performance indexes of the Catalyst Mixing Problem obtained by the CVP-DCT on 11 time grids for different *Ndctc*.

Ndctc	$\mathbf{J}^*$	***Ndctc***	$\mathbf{J}^*$	***Ndctc***	$\mathbf{J}^*$
2	0.045828	5	0.047788	8	0.047909
3	0.046959	6	0.047795	9	0.047964
4	0.047368	7	0.047862	10	0.047974

The optimum state and control trajectories of the solutions obtained with *Ntgrid*=11 are given on Figures 6.34 - 6.38.

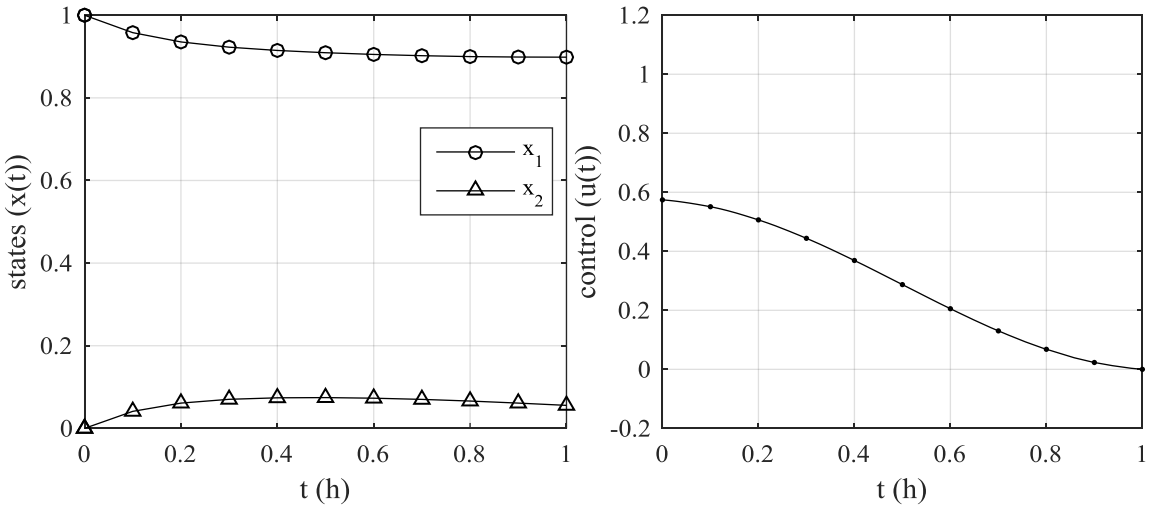


Figure 6.34 Optimal state and control trajectories of the Catalyst Mixing Problem obtained with *Ndctc*=2 and *Ntgrid*=11

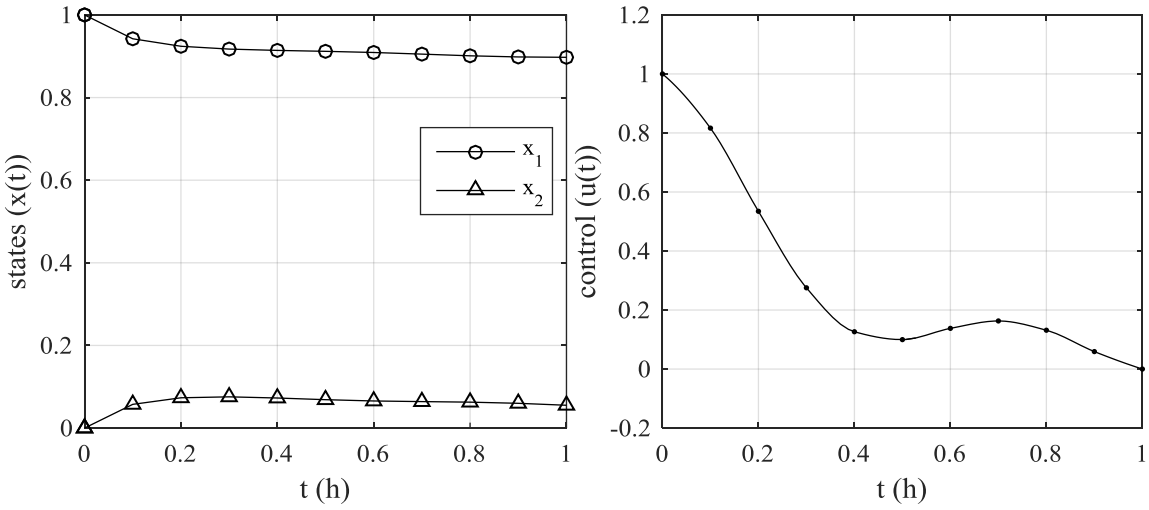


Figure 6.35. Optimal state and control trajectories of the Catalyst Mixing Problem obtained with *Ndctc*=4 and *Ntgrid*=11.

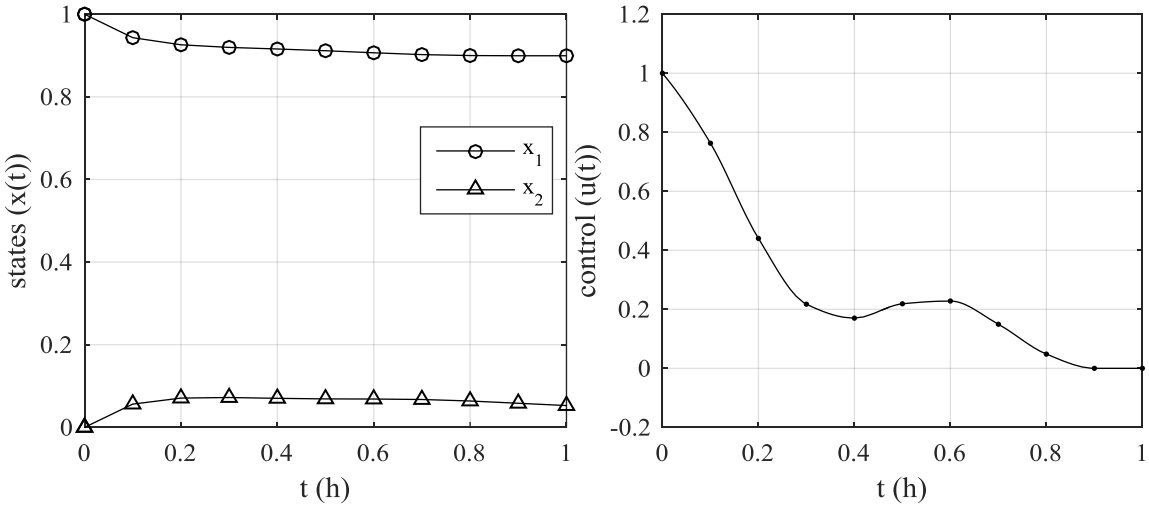


Figure 6.36. Optimal state and control trajectories of the Catalyst Mixing Problem obtained with *Ndctc*= 6 and *Ntgrid*=11.

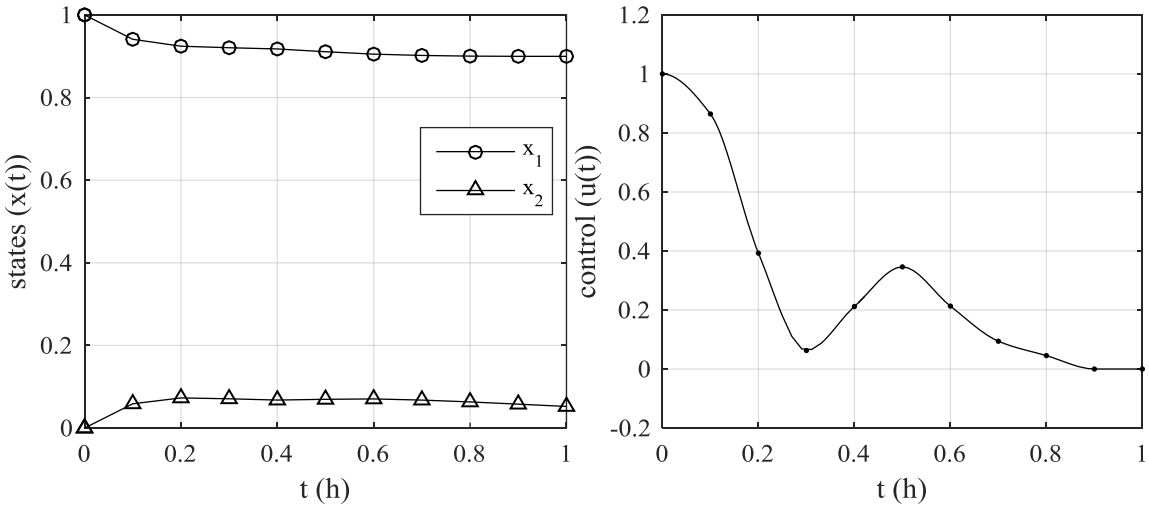


Figure 6.37. Optimal state and control trajectories of the Catalyst Mixing Problem obtained with *Ndctc*=8 and *Ntgrid*=11.

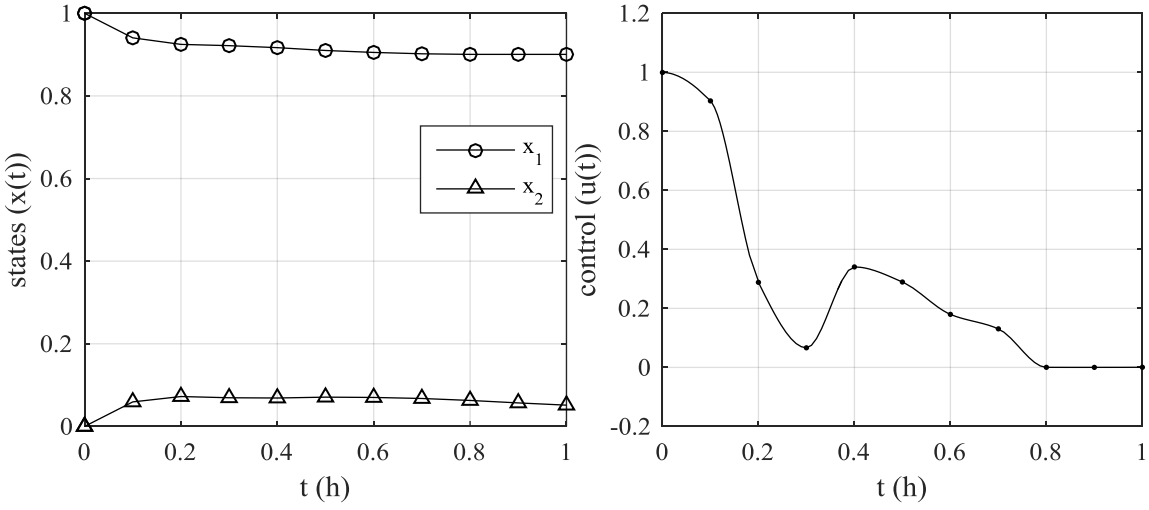


Figure 6.38. Optimal state and control trajectories of the Catalyst Mixing Problem obtained with *Ndctc*=10 and *Ntgrid*=11.

The SP function was added for *Ndctc*=10 and *Ntgrid*=11 case (Figure 6.39) and the performance index changes from 0.047974 to 0.047534 but with a smoother control profile

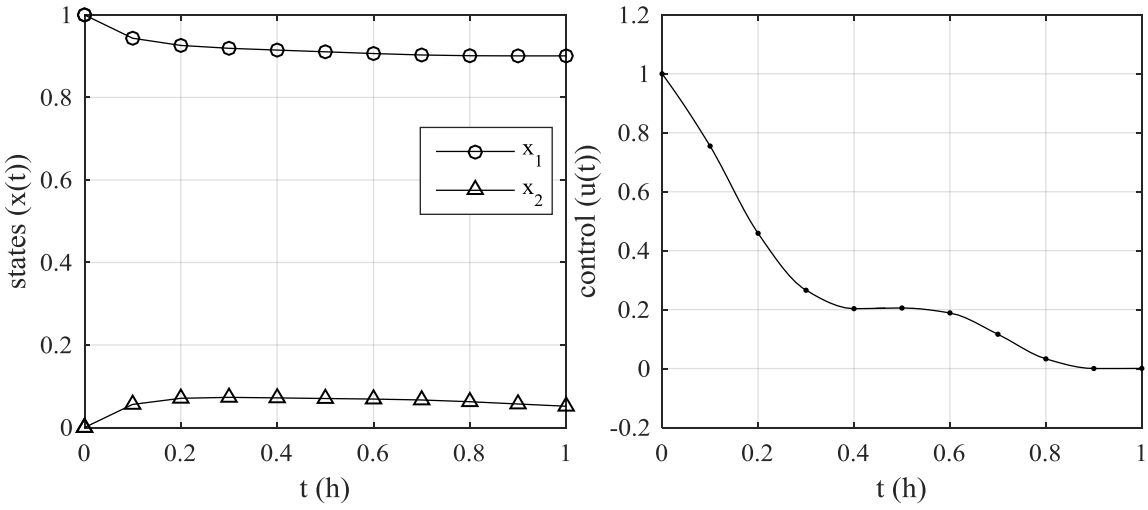


Figure 6.39. Optimal state and control trajectories of the Catalyst Mixing Problem obtained with *Ndctc*=10 and *Ntgrid*=11 with SP.

6.2.3. Comparison of the Solutions for the Catalyst Mixing Problem

This benchmark problem was solved in many publications. The reported optimum performance indexes of this problem are given in Table 6.12 in decreasing order. The best performance index calculated by Tanartkit and Biegler (1997) using the Nested Simultaneous Strategy and was reported as 0.04807. The CVO with *Ntgrid*=31 has the value closest to it. In the Table 6.12, the CVP-DCT solutions only with the lowest and the highest number of parameter are provided. The CVP-DCT with *Ndctc*=2 and *Ntgrid*=11 yields J^* = 0.045828 and this value deviates from the best reported solution by 4.66%. Taking *Ndctc*=10 and *Ntgrid*=11 yields J^* = 0.047974 and its deviation from best solution is 0.2%. For this problem, the CVP-DCT method worked well and provided a good approximation with only few parameters.

Table 6.12. Performance of different methods for the Catalyst Mixing Problem.

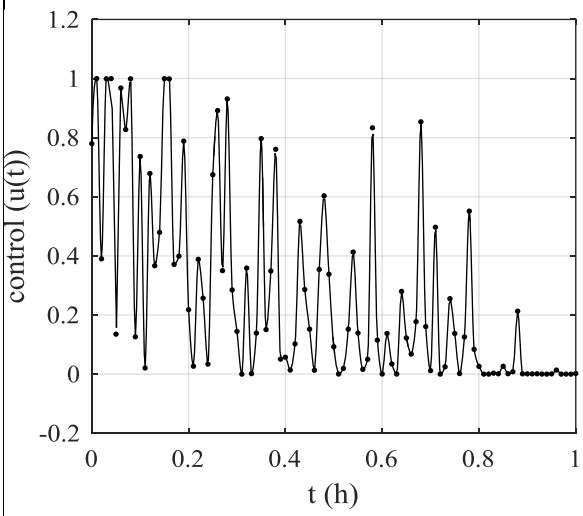
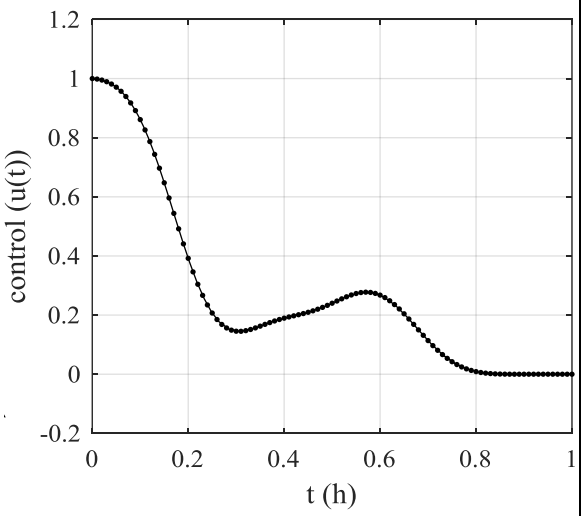
Method	**J^***	**Relative difference (%)**
Nested Simultaneous Strategy (Tanartkit and Biegler, 1997)	0.04807	-
CVO (*Ntgrid*= 31)	0.048063	0.015
Pseudospectral Collocation (PROPT) (Rutquist *et al*., 2010)	0.04805928	0.022
Control Parameterization (Vassiliadis, 1993)	0.0480557	0.030
Simultaneous Strategy (Liu *et al*., 2013)	0.0480557	0.030
Iterative Dynamic Programming (Dadebo and Mcauley, 1995)	0.04805	0.042
Direct Collocation (DIRCOL) (von Stryk, 1999)	0.048045599	0.051
CVP-DCT (*Ndctc*=10, *Ntgrid*=11)	0.047974	0.200
Piecewise Constant CVP (Hirmajer *et al*., 2009)	0.04796964	0.209
Integrated Controlled Random Search (ICRS) (Banga *et al*.,1998)	0.0479	0.354
Orthogonal Collocation on Finite Elements (DYNOPT) (Cizniar, *et al*., 2005)	0.0477457	0.675
CVP-DCT (*Ndctc*=2, *Ntgrid*=11)	0.045828	4.664

6.2.4. A Further Illustration of the Power of the CVP-DCT Method in Compressive Parameterization of the Control Trajectory for the Catalyst Mixing Problem

Numerical comparison of the CVO and the CVP-DCT method to solve the Catalyst Mixing Problem is provided in this section. Firstly, on the same time grids (*Ntgrid* = 101), CVP-DCT yielded a better performance index. Moreover, it used fewer (one tenth of the CVO) optimization decision variables. Comparison of the optimization statistics of both methods showed that "fmincon" required more function evaluations for CVO and this is mostly caused by the large dimension of the Hessian matrix. Furthermore, the DE algorithm could not find a solution within the desired tolerance limits because DE algorithm reached the maximum iteration and maximum functional evaluation limits set. With the CVO method, the ODE integrator was called 34 times more than it was called in the CVP-DCT

method. Finally, the required CPU time was 27.4 times longer in the CVO method. The comparison of computational statistics along with the corresponding control trajectories are given in Table. 6.13.

Table 6.13. Comparison via Computational Statistics of the CVO and CVP-DCT for the Catalyst Mixing Problem.

CVO with *Ntgrid*=101	CVP-DCT with *Ntgrid*=101, *Ndctc*=10
	
Number of decision variables : 101	Number of decision variables : 10
J* = 0.047162	J* = 0.047952
<u>"fmincon" stats</u> 47 iterations 9638 objective function evaluations	<u>"fmincon" stats</u> 21 iterations 4421 objective function evaluations
<u>DE stats</u> 2500 iterations 50000 objective function evaluations	<u>DE stats</u> 395 iterations 7900 objective function evaluations
Number of ODE function evaluations: 34,709,070	Number of ODE function evaluations: 1,011,819
CPU: 4033 seconds	CPU: 147 seconds

6.3.The Van der Pol Oscillator Problem

The Van der Pol Oscillator problem is formulated by Gritsis (1990) and studied by many authors. The optimal-control problem is given as follows.

$$\dot{x}_1 = (1 - x_2)\, x_1 - x_2 + u \tag{6.19}$$

$$\dot{x}_2 = x_1 \tag{6.20}$$

$$\dot{x}_3 = x_1^2 + x_2^2 + u^2 \tag{6.21}$$

$$\mathbf{x}(0) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{6.22}$$

Given these state ODEs and initial conditions, the upper and lower bounds of the control variable is given as,

$$-0.3 \leq u(t) \leq 1 \tag{6.23}$$

The performance index is to minimize x_3 at final time t_f which is equal to 5 hr. To convert this constrained problem to an unconstrained optimization problem the ULP Function given in Equation (4.8) was augmented to performance index. The penalty parameter (μ_{ul}) is taken as 10.

$$J = x_3(t_f) + ULP \tag{6.24}$$

6.3.1. Solution of the Van der Pol Oscillator Problem with the CVO Method

The problem was solved with *Ntgrid*=11 and *Ntgrid*=21 and the calculated J^* values are 2.86922 and 2.86947, respectively. It should be noted that for this problem fewer *Ntgrid* yields a better solution. $u(t_k) = 0$, k=1, ..., *Ntgrid*. .The interpolation method is selected as pchip and control variable bounds, Equation (6.23), are not violated. The optimum state and control trajectories are given on Figures 6.40 and 6.41.

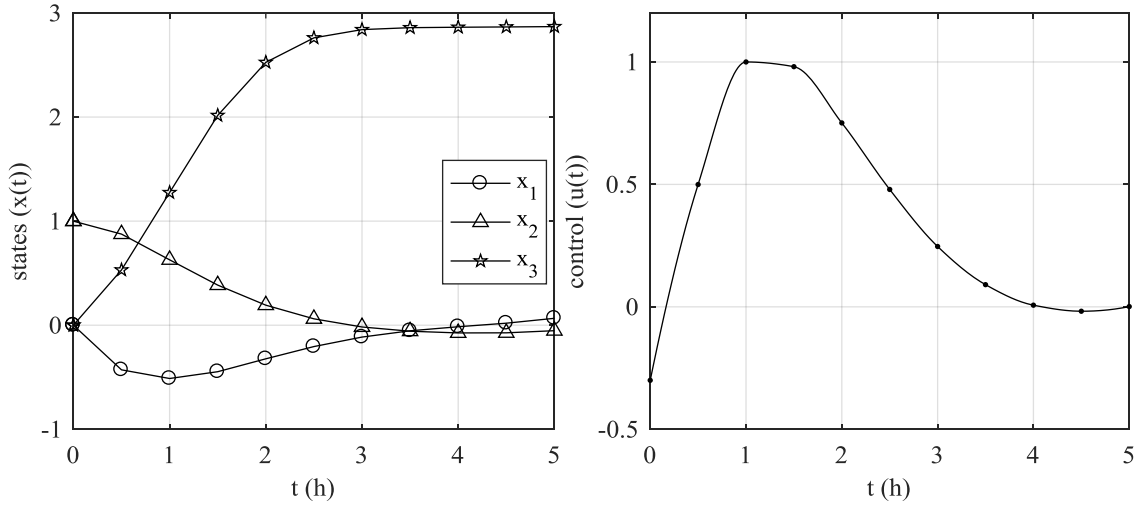


Figure 6.40. Optimal state and control trajectories of the Van der Pol Oscillator Problem via the CVO method for *Ntgrid*=11.

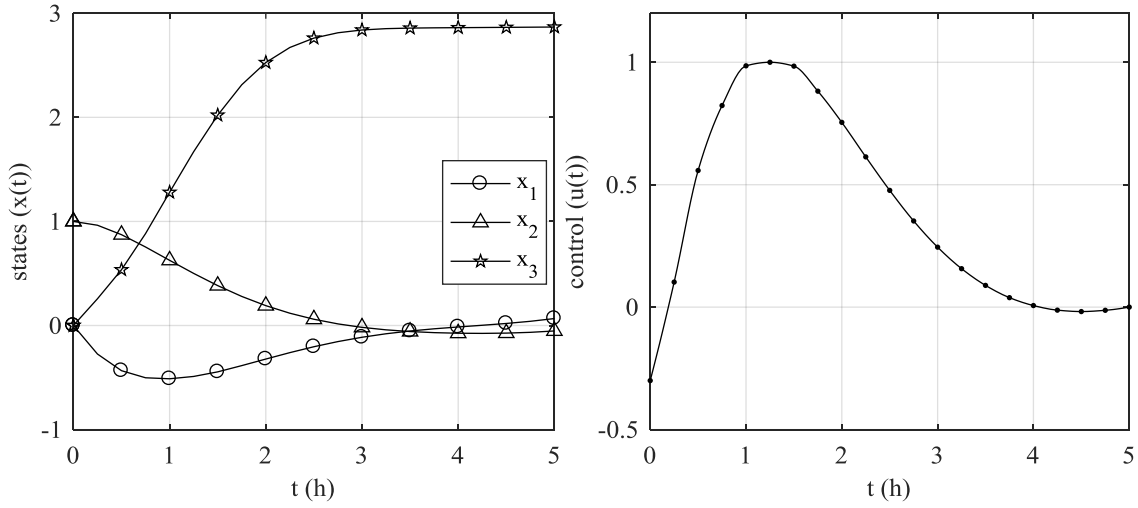


Figure 6.41. Optimal state and control trajectories of the Van der Pol Oscillator Problem via the CVO method for *Ntgrid*=21.

6.3.2. Solution of the Van der Pol Oscillator Problem with the CVP Method

The problem was solved with the CVP Method by using functional forms (v), (vi) and (vii) which are 3rd, 4th and 5th order polynomial functions. The initial guesses of all parameters were taken zero. The functional forms and the corresponding performance indexes are given in Table 6.14. Since the form of the control trajectory is like a 3rd or more order polynomial, the CVP method works well and increasing the polynomial order yields

better performance indexes. The optimum state and control trajectories are given on Figures 6.42, 6.43 and 6.44.

Table 6.14. Functional Forms used in the CVP to solve the Van der Pol Problem and the corresponding J* values.

Functional form	**J***
(v) 3rd order polynomial	2.914888
(vi) 4th order polynomial	2.877169
(vii) 5th order polynomial	2.873200

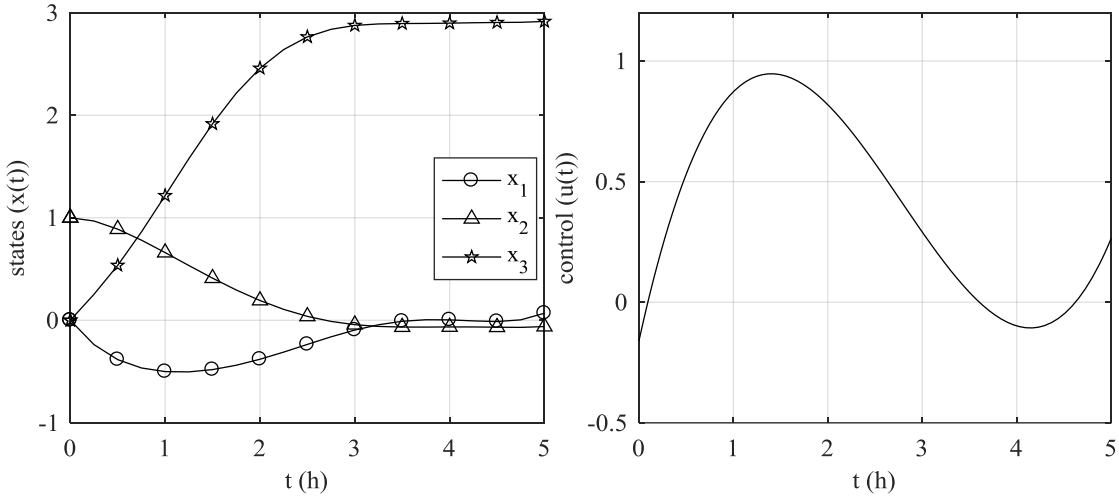


Figure 6.42. Optimal state and control trajectories of the Van der Pol Problem via the CVP Method with functional form (v).

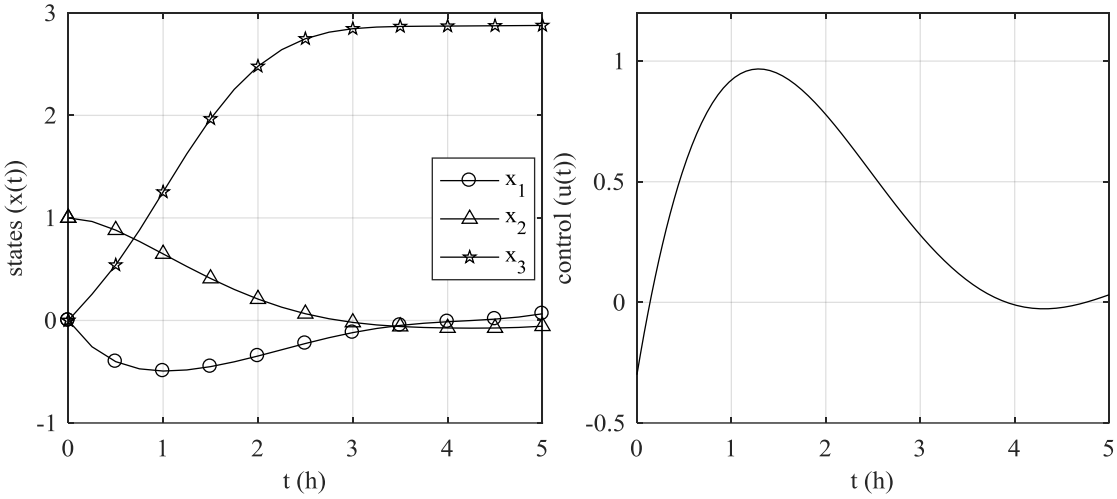


Figure 6.43. Optimal state and control trajectories of the Van der Pol Problem via the CVP Method with functional form (vi).

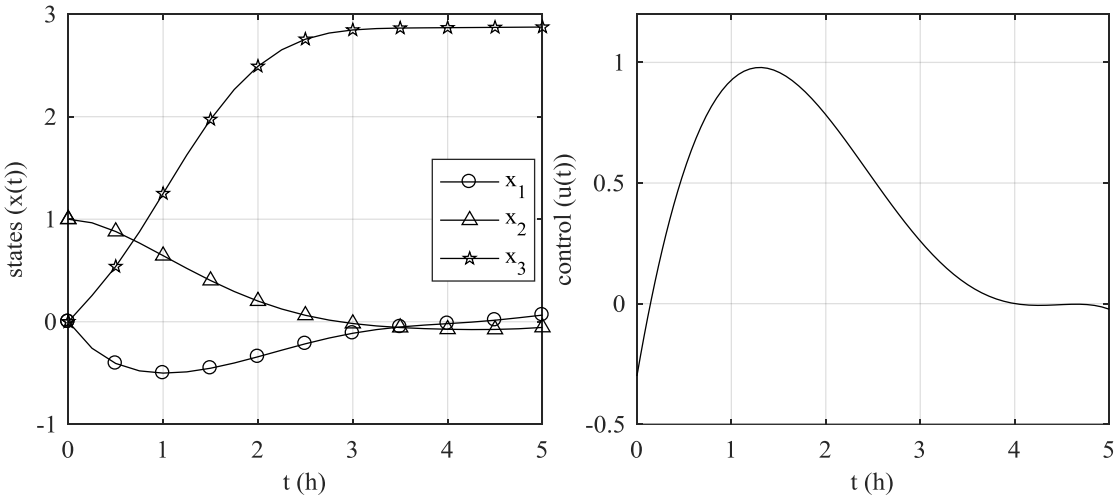


Figure 6.44. Optimal state and control trajectories to solve Van der Pol Problem via the CVP Method with functional form (vii).

6.3.3. Solution of the Van der Pol Oscillator Problem with the CVP-DCT Method

The problem is solved with *Ntgrid* = 11 by taking different *Ndctc*. The initial guess was selected as $\mathbf{u}_{\text{grid}} = 0$ and upper and lower bounds are set as -5 and +5 for "fmincon". The optimal performance indexes are given in Table 6.15.

Table 6.15. Optimal performance indexes of the Van der Pol Oscillator Problem by CVP-DCT on 11 time grids for different *Ndctc*.

Ndctc	**J***	***Ndctc***	**J***	***Ndctc***	**J***
2	4.496122	5	2.932456	8	2.868902
3	3.359986	6	2.890531	9	2.869191
4	2.989476	7	2.874191	10	2.865145

By checking the relative difference of optimum performance indexes obtained by the consecutive *Ndctc* values, it can be concluded that this problem requires more than 4 DCTCs to have a solution closer to the best solution in the literature which is reported by Morison (1991) as 2.867. After *Ndctc*=6 and *Ndctc*=7, the difference in the J* values becomes relatively small. It should be noted that the relative difference between the optimum performance indexes obtained by *Ndctc*=6 and *Ndctc*=10 is less than 1%. The optimum state and control trajectories by *Ndctc*=6 and *Ndctc*=10 are given by Figure 6.45 and 6.46.

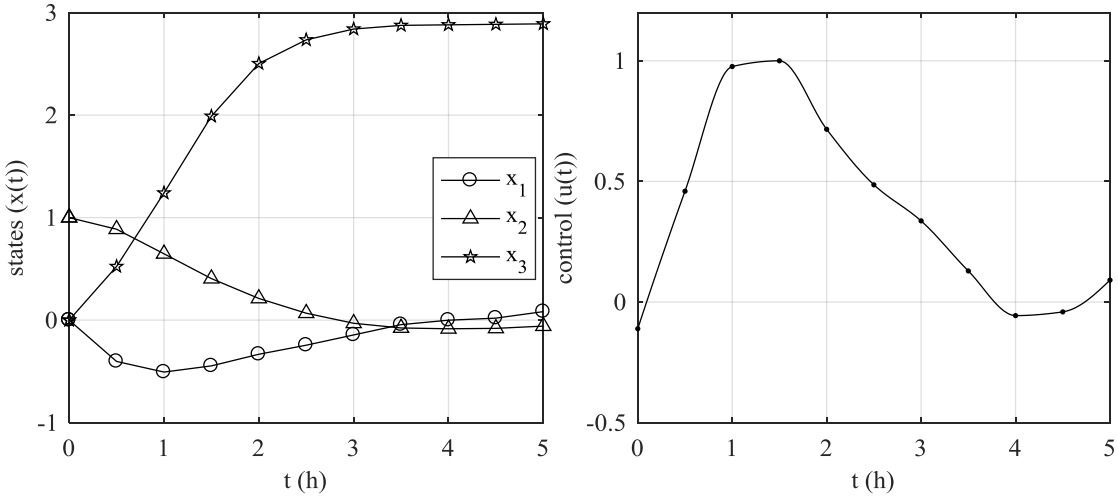


Figure 6.45. Optimal state and control trajectories of the Van der Pol Problem via CVP-DCT for *Ndctc*=6 and *Ntgrid*=11.

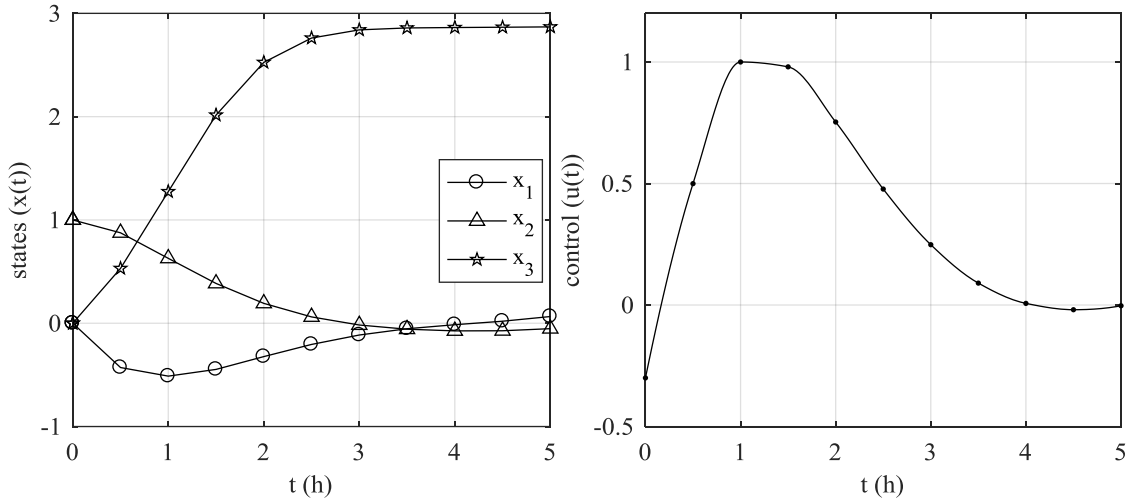


Figure 6.46 Optimal state and control trajectories of the Van der Pol Problem via CVP-DCT for *Ndctc*=10 and *Ntgrid*=11.

6.3.4. Comparison of the Solutions for the Van der Pol Oscillator Problem

The Van der Pol Oscillator problem is solved by many authors. The reported optimum performance indexes of this problem are given in Table 6.16 in increasing order. The best performance index among the reported solutions (J^*=2.867) is proposed by Morison (1984) using a sequential method. The CVP-DCT with *Ndctc*=10 and *Ntgrid*=11 has the value closest values to it. The relative percentage differences of each solution with the best-known solution is also represented in the Table 6.16. According to table, the solution with *Ndctc*=6 has less than 0.1% relative difference and this solution requires only six DCTCs. The best CVP-DCT solution obtained by *Ndctc*=10 which has 0.075% relative difference. This difference is getting higher by using lower *Ndctc*. With *Ndctc*=4, the optimum solution of CVP-DCT method has 3.94% of relative difference. Furthermore, the CVP solution with 5^{th} order polynomial (which uses six decision variables) yields a better performance index than solution with *Ndctc*=6. This is caused by the resemblance between the trajectories of the possible global optimum and a 5^{th} order polynomial.

Table 6.16. Performance of different methods for the Van der Pol Oscillator problem.

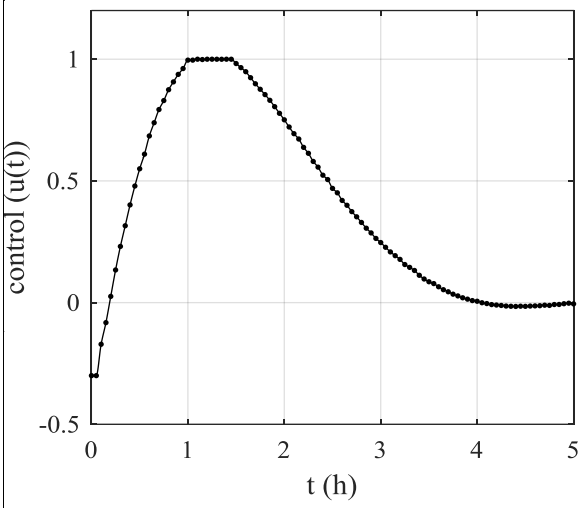
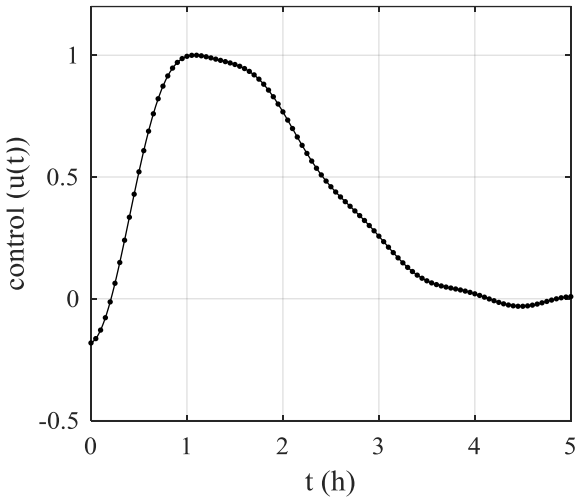
Method	**J^***	**Relative difference (%)**
Control Parameterization (Morison, 1984)	2.867	-
Pseudospectral Collocation (PROPT) (Rutquist *et al*., 2010)	2.86726	0.009
Control Parameterization (Gritsis, 1990)	2.868	0.035
Control Parameterization (Vassiliadis, 1993)	2.86809	0.038
CVP-DCT (*Ndctc*=10, *Ntgrid* = 11)	2.86915	0.075
CVP-DCT (*Ndctc*=9, *Ntgrid* = 11)	2.86919	0.076
CVO (*Ntgrid* = 11)	2.86922	0.077
Infeasible Path Approach (Tanartkit and Biegler, 1995)	2.8695	0.087
CVP with 5^{th} order polynomial	2.8732	0.216
CVP with 4^{th} order polynomial	2.87717	0.355
CVP-DCT (*Ndctc*=6, *Ntgrid* = 11)	2.89053	0.821
CVP with 3^{rd} order polynomial	2.91489	1.67
CVP-DCT (*Ndctc*=5, *Ntgrid* = 11)	2.93246	2.283
Indirect Method (Betts, 2010)	2.95373	3.025
Piecewise Constant CVP (Hirmajer *et al*., 2009)	2.961	3.279
TPBVP (Bell and Sargent, 2000)	2.9758	3.795
CVP-DCT (*Ndctc*=4, *Ntgrid* = 11)	2.98948	3.941

6.3.5. A Further Illustration of the Power of the CVP-DCT Method in Compressive Parameterization of the Control Trajectory for the Van der Pol Problem

In this final section, we numerically compare the solutions obtained by CVO with *Ntgrid* = 101 and CVP-DCT with *Ndctc* = 10 and *Ntgrid* = 101. For this problem, the CVO method gave a better optimum performance index and unlike the previous example problems, the optimum control trajectory had a smooth profile with *Ntgrid* = 101. Solution with CVP-DCT was very close to CVO's solution. The difference between those

performance indexes was 0.0046 (0.001% relative difference). However, the CVP-DCT method required 10 decision variables whereas CVO had 101 decision variables. Furthermore, comparison of the "fmincon" statistics showed that the solution obtained by CVP-DCT required one-tenth of the functional evaluations that CVO required. Furthermore, CVO returned a better solution with the DE algorithm even though it could not find a solution within the desired tolerance limits due to reaching the maximum iteration and maximum function evaluation limits set. With the CVO method, the ODE integrator was called 35 times more than it was called by the CVP-DCT method. Finally, the required CPU time was 27.4 times longer in the CVO method. The comparison of the computational statistics along with the corresponding control trajectories are given in Table. 6.17.

Table 6.17. Comparison via Computational Statistics of the CVO and CVP-DCT for the Van der Pol Oscillator Problem.

CVO with *Ntgrid*=101	CVP-DCT with *Ntgrid*=101, *Ndctc*=10
	
Number of decision variables : 101	Number of decision variables : 10
J* = 2.8672	J* = 2.8718
"fmincon" stats 26 iterations 5559 objective function evaluations	"fmincon" stats 20 iterations 535 objective function evaluations
DE stats 2500 iterations 50000 objective function evaluations	DE stats 533 iterations 10660 objective function evaluations
Number of ODE function evaluations: 29,374,016	Number of ODE function evaluations: 2,430,028
CPU: 5279 seconds	CPU: 569 seconds

6.4. The Consecutive Batch Reaction Problem

The problem consists of a consecutive reaction in an adiabatic batch reactor in which the temperature is exactly controlled. The problem belongs to Ray (1981). The reaction

formula is $A \rightarrow B \rightarrow C$, and it is 2nd order with respect to the first step and 1st order with respect to the second step.

The optimal-control problem consists of two ODEs which are material balances of the species A and B. The system model with the initial conditions are given as follows.

$$\dot{x}_1 = -k_1 x_1^2 \tag{6.25}$$

$$\dot{x}_2 = k_1 x_1^2 - k_2 x_2 \tag{6.26}$$

$$\mathbf{x}(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.27}$$

x_1 and x_2 represent the concentration of species A and B, respectively. Rate constants, k_1 and k_2, are defined by the Arrhenius Equations:

$$k_1 = 4000\, e^{-5000/(1.9872\ u)} \tag{6.28}$$

$$k_2 = 62000\, e^{-10000/(1.9872\ u)} \tag{6.29}$$

The control variable, u, is the reactor temperature which is bounded as:

$$298 \leq u(t) \leq 398 \tag{6.30}$$

The performance index is to maximize x_2 (therefore to minimize $-x_2$) at the final-time t_f which is equal to 1 hr. To convert this constrained problem to an unconstrained optimization problem, the ULP Function given in Equation (4.8) was used. The penalty parameter (μ_{ul}) was taken as 10. The augmented performance index became:

$$J = -x_2(t_f) + ULP \tag{6.31}$$

6.4.1. Solution of the Consecutive Batch Reaction Problem with the CVO Method

The problem was solved with *Ntgrid*= 11 and *Ntgrid*=21 and the calculated J^* values are 0.61072 and 0.61079 respectively. The initial guess was taken as $u(t_k) = 348$, k=1, ..., *Ntgrid*. The interpolation method was selected as spline. Higher *Ntgrid* yielded a better

performance index. However, it required more function evaluations. Control variable bounds, Equation (6.30) were not violated. The optimum state and control trajectories are given on Figures 6.47 and 6.48.

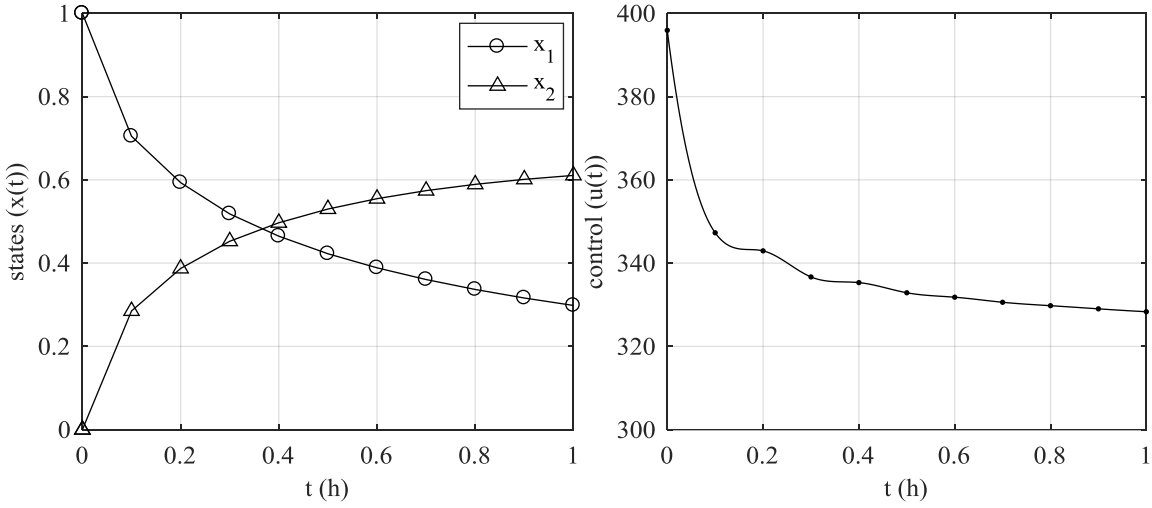


Figure 6.47. Optimal state and control trajectories of the Consecutive Batch Reaction Problem via CVO for *Ntgrid*=11.

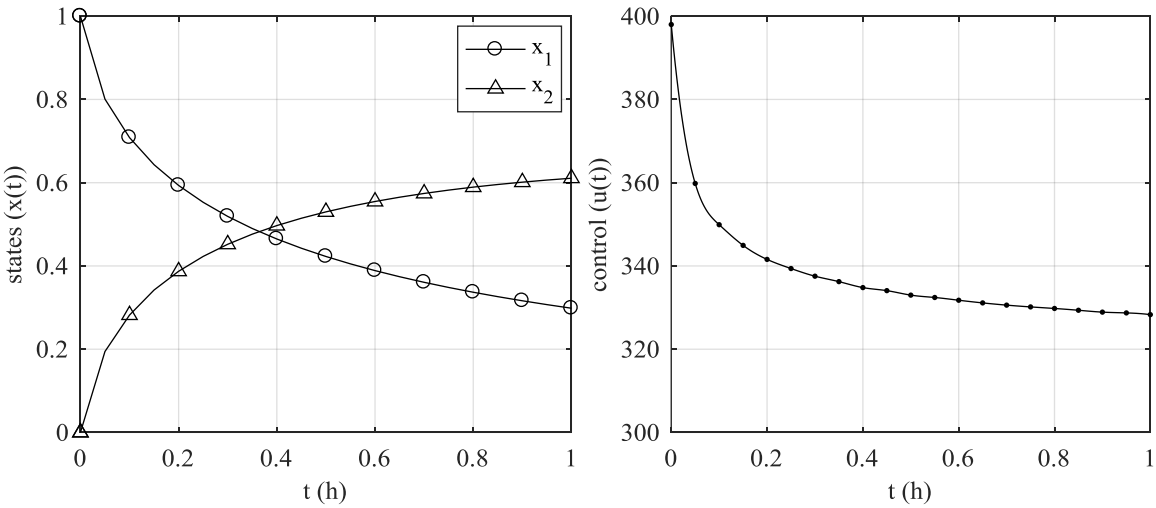


Figure 6.48 Optimal state and control trajectories of the Consecutive Batch Reaction Problem via CVO for *Ntgrid*=21.

6.4.2. Solution of the Consecutive Batch Reaction Problem with the CVP Method

This problem was also solved by the CVP method with seven functional forms discussed in Chapter 4. All functional forms have a constant parameter and initial guess of that parameter was taken as 348, rest of the parameters were taken as zero. The optimum performance indexes obtained are tabulated in Table 6.18. Functional form (iii) worked well and yielded the best performance indexes among other forms. Furthermore, functional form (ii) yielded a very close solution to (iii) with fewer parameters. The optimum state and control trajectories of the solutions are identical and given in Figure 6.49.

Table 6.18. Functional forms used in the CVP to solve the Consecutive Batch Reaction Problem and the corresponding J* values.

Functional form	**J***
(i) Constant + exponential function	0.60889
(ii) Constant + exponential term	0.61054
(iii) Constant + 1^{st} order polynomial + exponential term	0.61071
(iv) 2^{nd} order polynomial	0.60988
(v) 3^{rd} order polynomial	0.61028
(vi) 4^{th} order polynomial	0.61048
(vii) 5^{th} order polynomial	0.61063

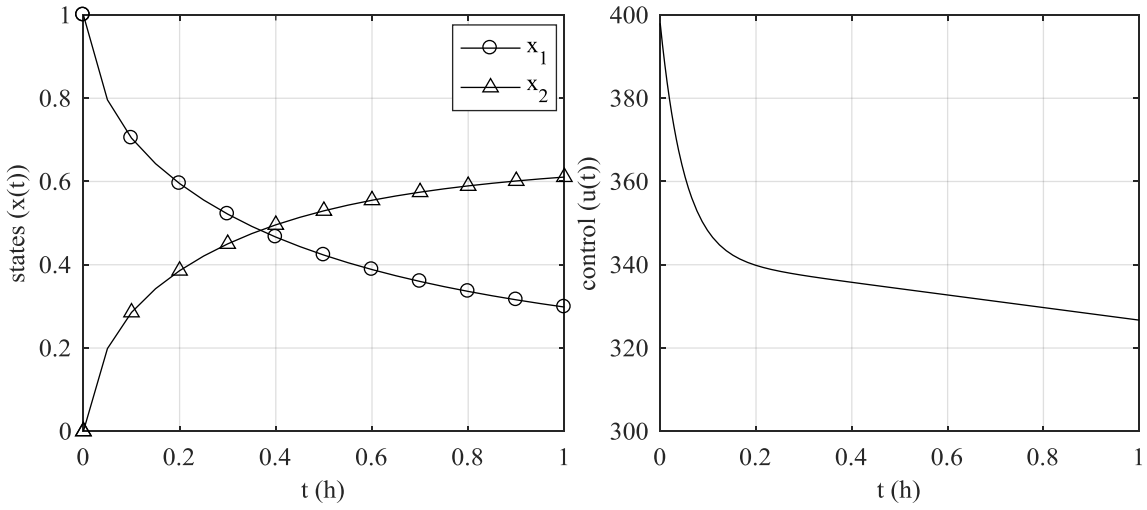


Figure 6.49. Optimal state and control trajectories of Consecutive Batch Reaction Problem via CVP with functional form (ii and iii).

6.4.3. Solution of the Consecutive Batch Reaction Problem with the CVP-DCT Method

The problem was solved with *Ntgrid* = 11 by using different *Ndctc* values. The initial guess was taken as DCT of constant trajectory of $u(t_k) = 348$, k=1, ..., *Ntgrid*. The upper and lower bounds on the decision variables (DCTCs) were set as -1000 and +1000 for "fmincon". The optimal performance indexes are given in Table 6.19.

Table 6.19. Optimal performance indexes of the Consecutive Batch Reaction Problem by CVP-DCT on 11 time grids for different *Ndctc* values.

Ndctc	**J***	***Ndctc***	**J***	***Ndctc***	**J***
2	0.60878	5	0.61014	8	0.61053
3	0.60949	6	0.61030	9	0.61063
4	0.60990	7	0.61042	10	0.61070

For this example, optimum performance indexes were very close to each other. In fact, the difference between the solution of *Ndctc*=10 and *Ndctc*=2 was quite low. Performance index calculated with *Ndctc* = 2 was only 0.31% worse than the performance

index calculated with *Ndctc*=10. The optimum state and control trajectories by *Ndctc* = 2 *Ndctc*=3, *Ndctc*=5, *Ndctc*=7 and *Ndctc* = 10 are given by Figures 6.50 – 6.54.

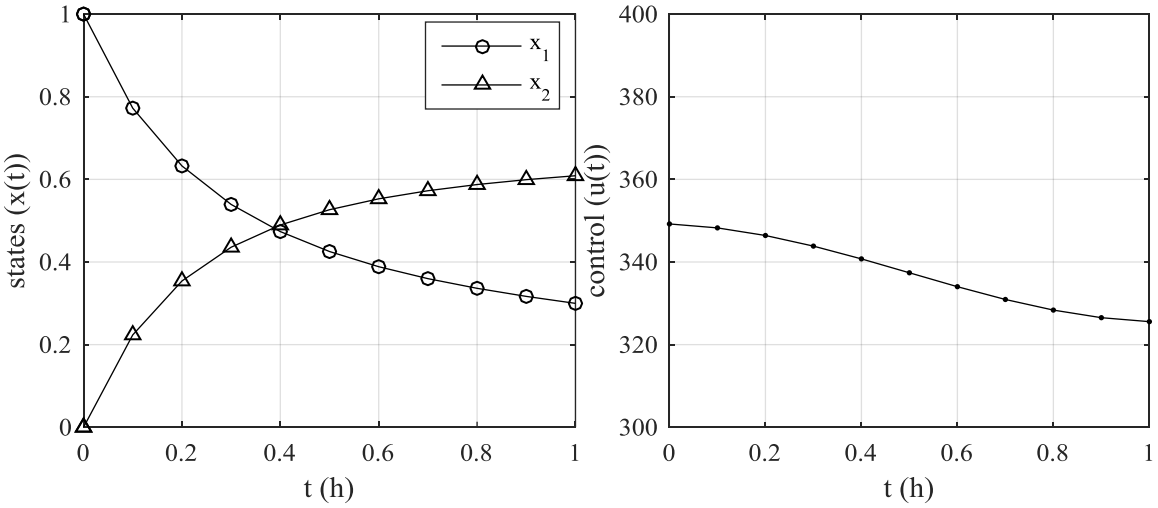


Figure 6.50 Optimal state and control trajectories of the Consecutive Batch Reaction Problem via CVP-DCT for *Ndctc*=2 and *Ntgrid*=11.

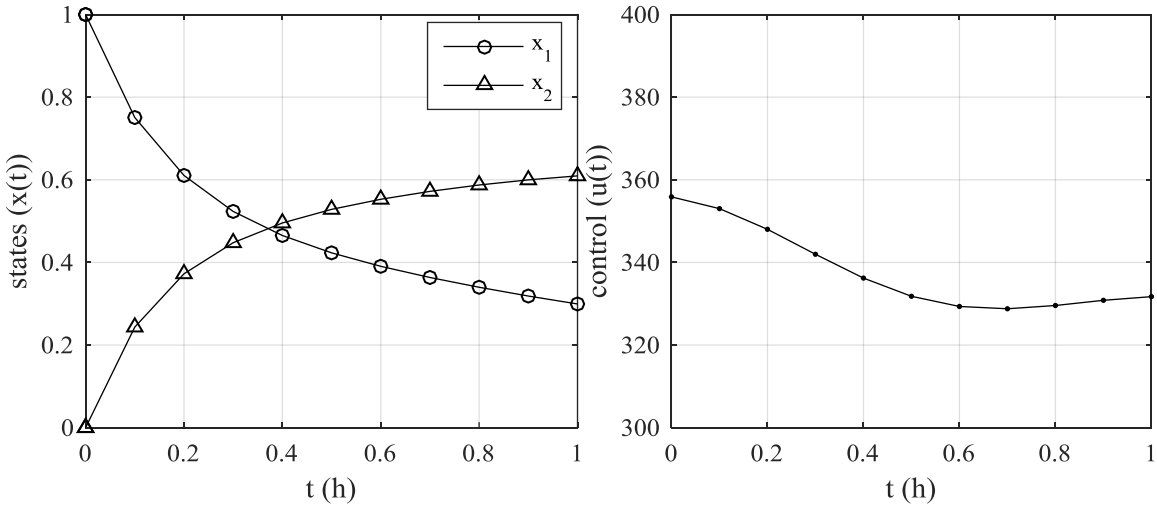


Figure 6.51. Optimal state and control trajectories of the Consecutive Batch Reaction Problem via CVP-DCT for *Ndctc*=3 and *Ntgrid*=11.

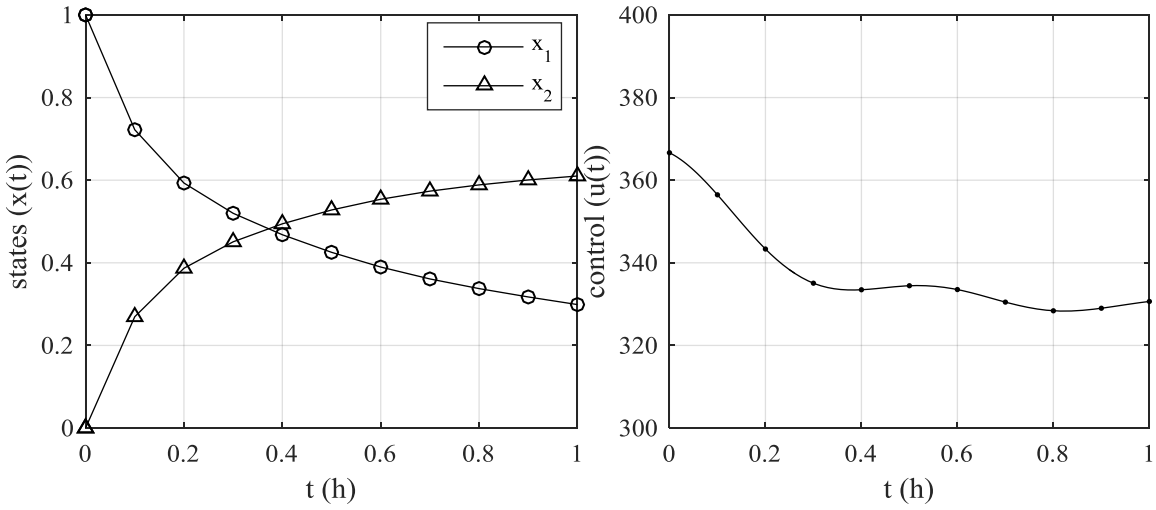


Figure 6.52. Optimal state and control trajectories of the Consecutive Batch Reaction Problem via CVP-DCT for *Ndctc*=5 and *Ntgrid*=11.

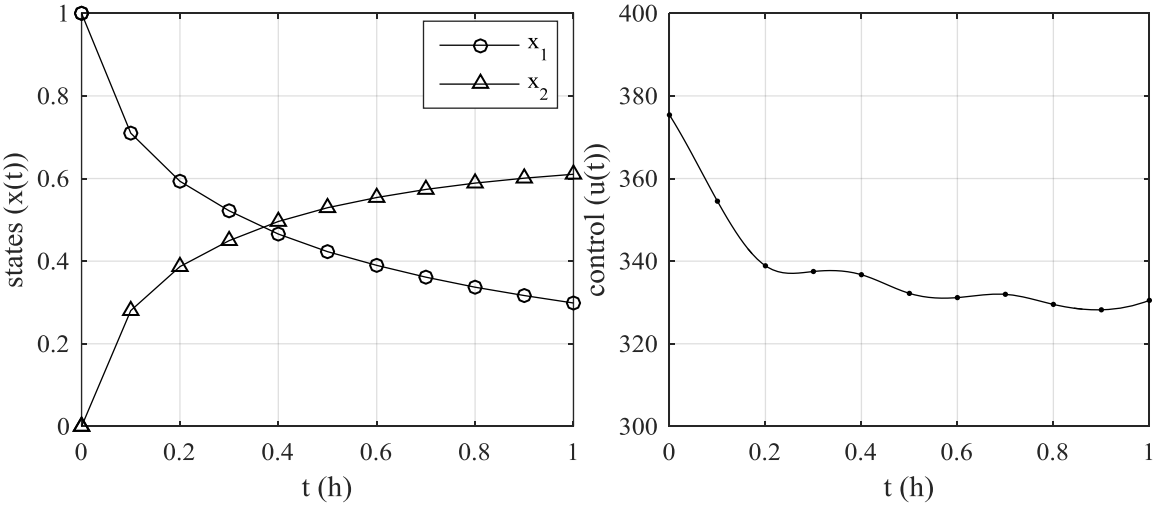


Figure 6.53. Optimal state and control trajectories of the Consecutive Batch Reaction Problem via CVP-DCT for *Ndctc*=7 and *Ntgrid*=11.

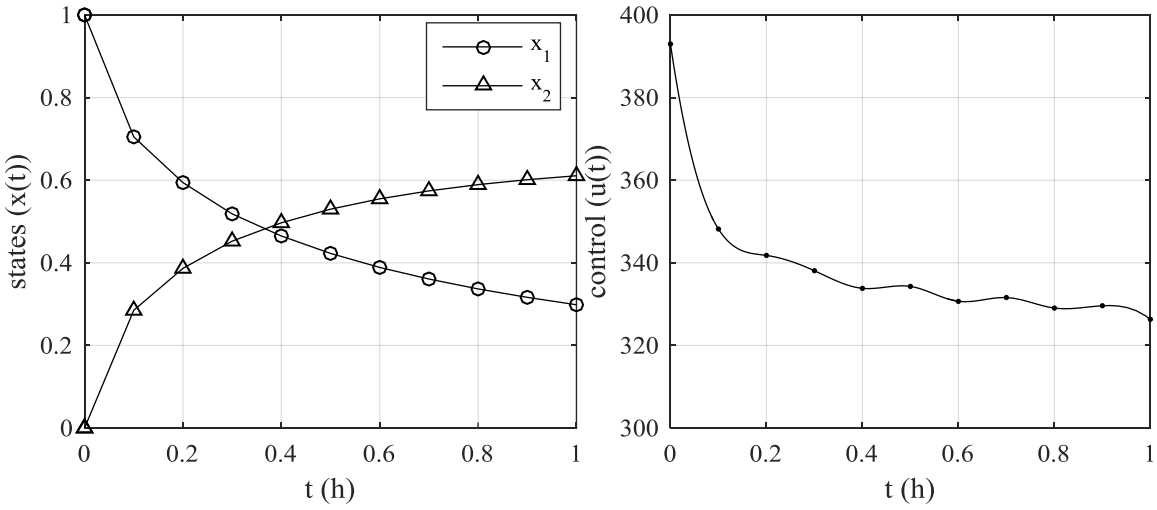


Figure 6.54. Optimal state and control trajectories of the Consecutive Batch Reaction Problem via CVP-DCT for *Ndctc*= 10 and *Ntgrid*=11.

6.4.4. Comparison of the Solutions for the Consecutive Batch Reaction Problem

The problem was solved by several authors. The best performance index among the reported solutions (J*=0.6108) belongs to PROPT solver of Tomlab Optimization Inc. (Rutquist *et al*, 2010) where the Pseudospectral Orthogonal Collocation Method is utilized. The reported solutions are compared in Table 6.20. Among the methods used in this thesis work, CVO with *Ntgrid* = 21 had the value closest to it with a 0.0016% relative difference. The solution obtained by the CVP-DCT method with *Ndctc*= 2 had only 0.3307% relative difference and it should be noted that the number of optimization decision variables was only two in that case. Furthermore, the performance indexes obtained with the same number of parameters by CVP and CVP-DCT were also compared. With four, five and six decision variables, CVP found better performance indexes than CVP-DCT.

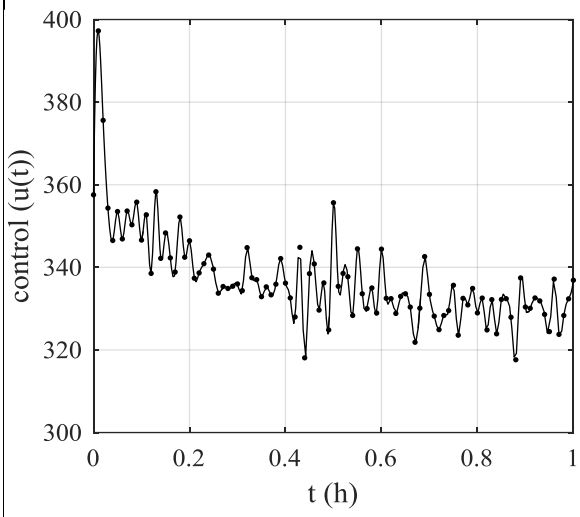
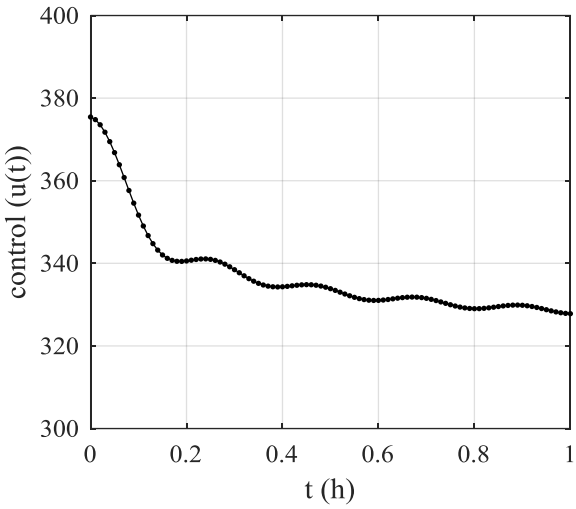
Table 6.20. Performance of different methods for the Consecutive Batch Reaction Problem.

Method	**J***	**Relative difference (%)**
Pseudospectral Collocation (PROPT)(Rutquist *et al*., 2010)	0.61080	-
CVO (with *Ntgrid*=21)	0.61079	0.0016
IDP (Dadebo and Mcauley, 1995; Logsdon and Biegler, 1989)	0.610775	0.0041
Direct Collocation (Logsdon and Biegler, 1989)	0.610767	0.0054
CVO (with *Ntgrid*=11)	0.61072	0.0131
CVP with functional form (iii)	0.61071	0.0147
CVP-DCT (*Ndctc*=10, *Ntgrid* = 11)	0.61070	0.0164
CVP with 5^{th} order polynomial	0.61063	0.0278
Orthogonal Collocation on Finite Elements (DYNOPT) (Cizniar *et al*., 2005)	0.610589	0.0345
CVP with 4^{th} order polynomial	0.61048	0.0524
Sequential with Ant Colony Optimization (Rajesh *et al*., 2001)	0.61045	0.0573
CVP-DCT (*Ndctc*=6, *Ntgrid* = 11)	0.6103	0.0819
CVP with 3^{rd} order polynomial	0.61028	0.0851
CVP-DCT (*Ndctc*=5, *Ntgrid* = 11)	0.61014	0.1081
Orthogonal Collocation (Renfro *et al*., 1987)	0.61	0.131
CVP-DCT (*Ndctc*=4, *Ntgrid* = 11)	0.6099	0.1473
CVP-DCT (*Ndctc*=2, *Ntgrid* = 11)	0.60878	0.3307

6.4.5. A Further Illustration of the Power of the CVP-DCT Method in Compressive Parameterization of the Control Trajectory for the Consecutive Batch Reaction Problem

In this final section, we numerically compare the solutions obtained by CVO with *Ntgrid*=101 and CVP-DCT with *Ndctc*=10 and *Ntgrid*=101. The solution with CVO required extensive computational effort and could not provide a stable solution. Therefore, the maximum iteration limits of the DE algorithm was taken as 3500 to get a solution. On the same time grids (*Ntgrid*=101), CVP-DCT yielded a better performance index. Moreover, it used fewer (one tenth of the CVO) optimization decision variables. With the CVO method, the ODE integrator was called 35 times more than it was called by the CVP-DCT method. Finally, the required CPU time was 19 times longer in the CVO method. The comparison of the computational statistics along with the corresponding control trajectories are given in Table 6.21.

Table 6.21. Comparison of CVO and CVP-DCT via Computational Statistics for Consecutive Batch Reaction Problem.

CVO with *Ntgrid*=101	CVP-DCT with *Ntgrid*=101, *Ndctc*=10
	
Number of decision variables : 101	Number of decision variables : 10
J* = 0.60933	J* = 0.61046
"fmincon" stats 8 iterations 1815 objective function evaluations	"fmincon" stats 5 iterations 274 objective function evaluations
DE stats 3500 iterations 70000 objective function evaluations	DE stats 211 iterations 4220 objective function evaluations
Number of ODE function evaluations: 37,842,495 CPU: 5613 seconds	Number of ODE function evaluations: 1,169,292 CPU: 295 seconds

6.5.The Plug-Flow Reactor Temperature Control Problem

The problem was proposed by Ko and Steve (1971). An exothermic, irreversible, first-order reaction A $\leftrightarrow$ B takes place in a plug-flow reactor and the reaction formula is given as follows.

The optimal-control problem consist of two ODEs which are material and energy balances and are given with the initial conditions as follows.

$$\dot{x}_1 = (1 - x_1)k_1 - x_1 k_2 \tag{6.32}$$

$$\dot{x}_2 = 300[(1 - x_1)k_1 - x_1 k_2] - u(x_2 - 290) \tag{6.33}$$

$$\mathbf{x}(0) = \begin{bmatrix} 0 \\ 380 \end{bmatrix} \tag{6.34}$$

x_1 and x_2 represent the normalized concentration of species A and the temperature in the reactor respectively. Rate constants, k_1 and k_2, are defined by the Arrhenius Equations:

$$k_1 = 1.7536 \times 10^5 \, e^{\frac{-1.1374 \times 10^4 /}{1.9872 \, x_2}} \tag{6.35}$$

$$k_2 = 2.4885 \times 10^{10} \, e^{\frac{-2.2748 \times 10^4}{1.9872 \, x_2}} \tag{6.36}$$

The control variable, u, is the normalized coolant flow rate which is given as:

$$0 \leq u(t) \leq 0.5 \tag{6.37}$$

Furthermore, a state-path inequality constraint which is an upper limit for reactor temperature for all times is also set as:

$$x_2(t) \leq 460 \qquad \forall \, t \in [0, 5] \tag{6.38}$$

The performance index is to maximize x_1 (therefore to minimize $-x_1$) at the final time t_f which is equal to 5 hr. To convert this constrained problem to an unconstrained optimization problem, the ULP Function given in Equation (4.8) was used. The penalty parameter (μ_{ul}) was taken as 10. Moreover, the path constraint, Equation (6.38), was handled by the Path-Constraint Penalty Function (PCP) given by Equation (6.39). The path-constraint penalty parameter, μ_{pc}, was taken as 100 for this example.

$$\mathrm{PCP} = \sum_{\mathrm{j}=1}^{Ntgrid} \mu_{\mathrm{pc}} max\{0,\ \mathrm{x}(\mathrm{t_j}) - 460\} \tag{6.39}$$

Both the ULP and PCP were added to the objective functional, hence the performance index became:

$$\mathrm{J} = -\mathrm{x_1}(\mathrm{t_f}) + \mathrm{ULP} + \mathrm{PPC} \tag{6.40}$$

6.5.1. Solution of the Plug Flow Reactor Problem with the CVO Method

The problem was solved with *Ntgrid*=11 and *Ntgrid*=21 and the calculated J^* values are 0.67502 and 0.67671, respectively. The initial guess was taken $u(t_k) = 0.25$, k=1, ..., *Ntgrid*. The interpolation method was selected as pchip. Control variable bounds, Equation (6.37), and inequality path constraint Equation (6.39) were not violated. The optimum state and control trajectories are given on Figures 6.55 and 6.56.

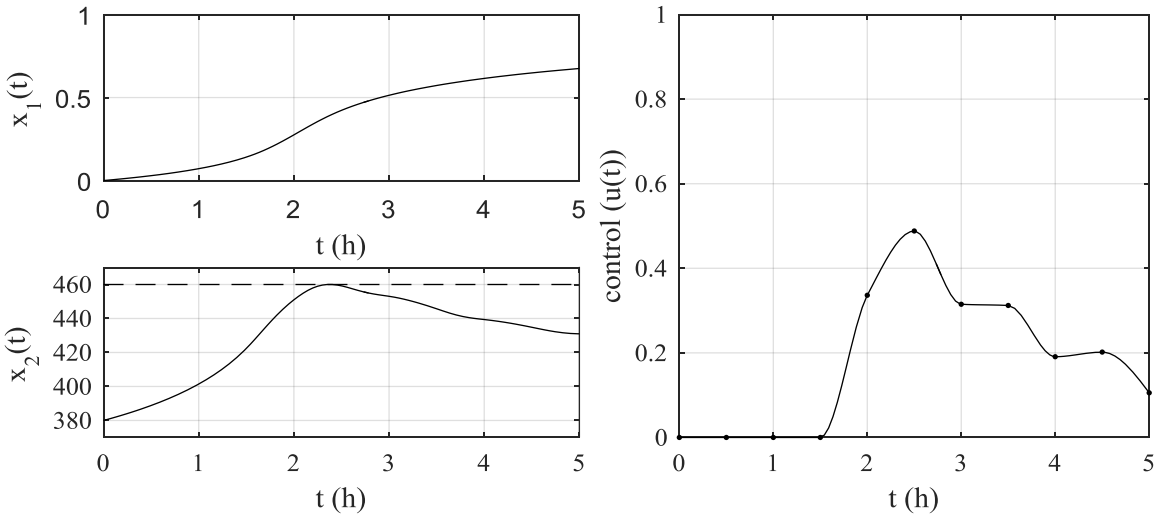


Figure 6.55. Optimal state and control trajectories of the Plug-Flow Reactor Temperature Control Problem via CVO for *Ntgrid*=11.

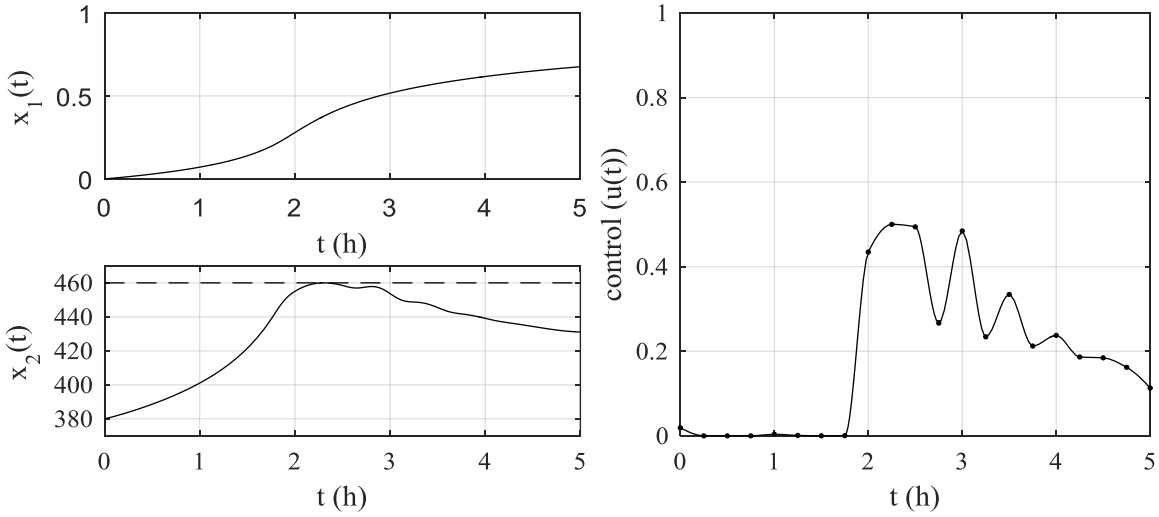


Figure 6.56. Optimal state and control trajectories of the Plug-Flow Reactor Temperature Control Problem via CVO for *Ntgrid*=21.

6.5.2. Solution of the Plug Flow Reactor Problem with the CVP Method

This problem was also solved by the CVP method with several functional forms. None of these functional forms yielded a solution close to CVO with *Ntgrid*=11. The performance index by the 3rd order polynomial, i.e. functional form (v), yielded an optimum solution of $J^* = 0.65893$. Higher order polynomials were expected to yield better solutions theoretically. However, "fmincon" cannot even decrease the objective functional to a single-digit number and the DE algorithm could not find a good solution with the default optimization parameters adopted in this thesis (TolX, CR, F, NP etc.), which had been discussed at the beginning of Chapter 6. Changing these parameters, e.g. using higher NP or tighter tolerance limits is computationally expensive and still does not guarantee better solutions. The optimum state and control trajectories obtained by the 3rd order polynomial are given in Figure 6.57.

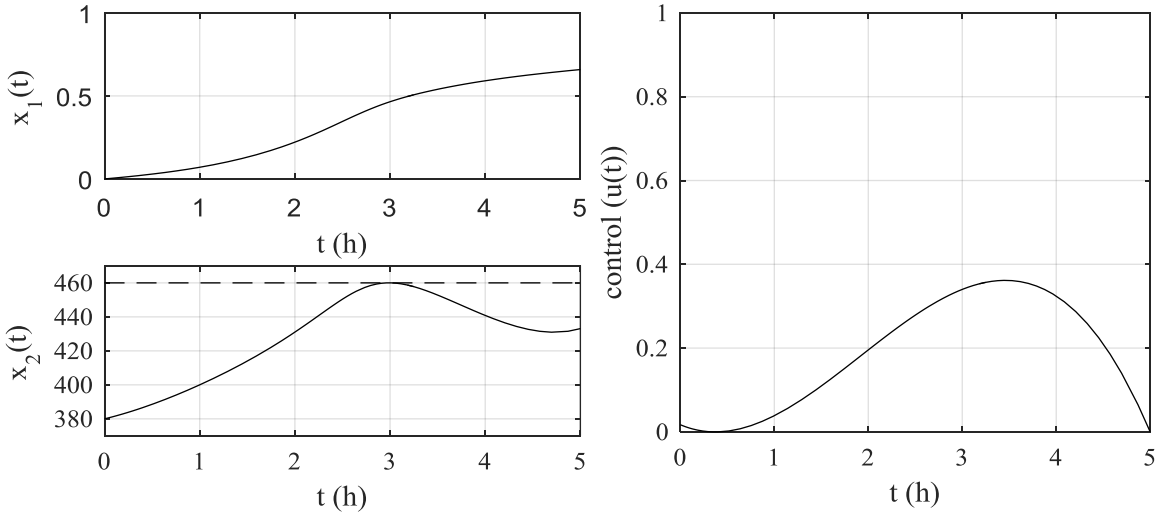


Figure 6.57. Optimal state and control trajectories of the Plug-Flow Reactor Temperature Control Problem via CVP with functional form (v).

6.5.3. Solution of the Plug Flow Reactor Problem with the CVP-DCT Method

The problem was solved with *Ntgrid*=11 by using different *Ndctc* values. The initial guess was selected DCT of constant trajectory of $u(t_k) = 0$, k=1, ..., *Ntgrid*, and the upper and lower bounds on the decision variables (DCTCs) were set as -10 and +10 for "fmincon". The optimal performance indexes are given in Table 6.22.

Table 6.22. Optimal performance indexes of the Plug-Flow Reactor Temperature Control Problem by CVP-DCT on 11 time grids for different *Ndctc*.

Ndctc	**J***	***Ndctc***	**J***	***Ndctc***	**J***
2	0.64193	5	0.66732	8	0.67181
3	0.64634	6	0.67095	9	0.67440
4	0.66487	7	0.67136	10	0.67461

With four parameters, the CVP-DCT method found a better performance index than the CVP method. If *Ndctc* ≥ 4 is used, the relative differences of the optimum performance indexes obtained by the consecutive *Ndctc* values are less than 0.5% and these differences decrease with increasing *Ndctc*. It should be noted that the relative difference between

Ndctc=4 and *Ndctc*=10 is only 1.44%. The optimum state and control trajectories by *Ndctc*=4, *Ndctc*=7, and *Ndctc*=10 are given by Figure 6.58, 6.59 and 6.60.

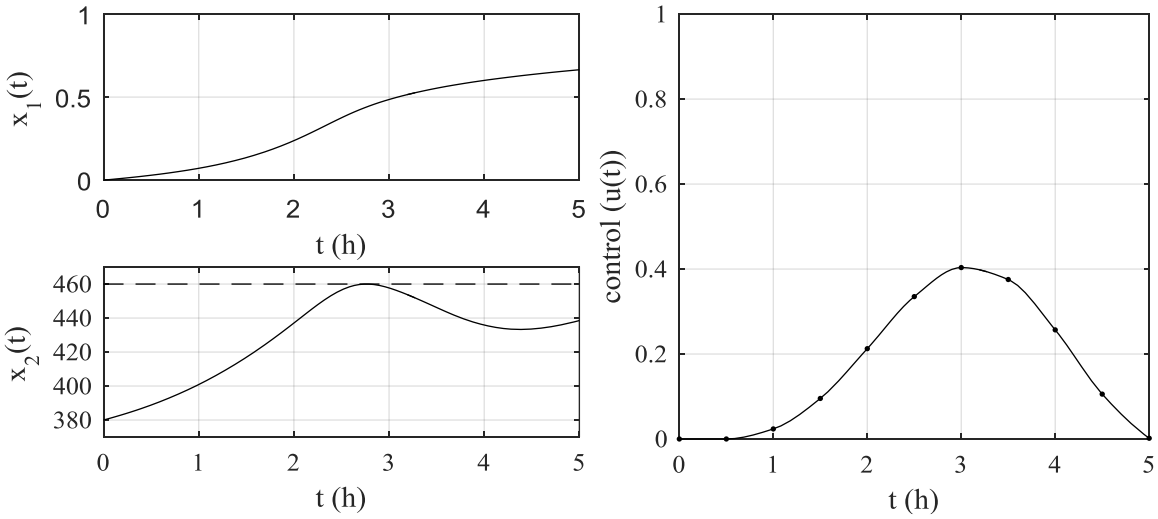


Figure 6.58. Optimal state and control trajectories of the Plug-Flow Reactor Temperature Control Problem via CVP-DCT for *Ndctc*=4 and *Ntgrid*=11.

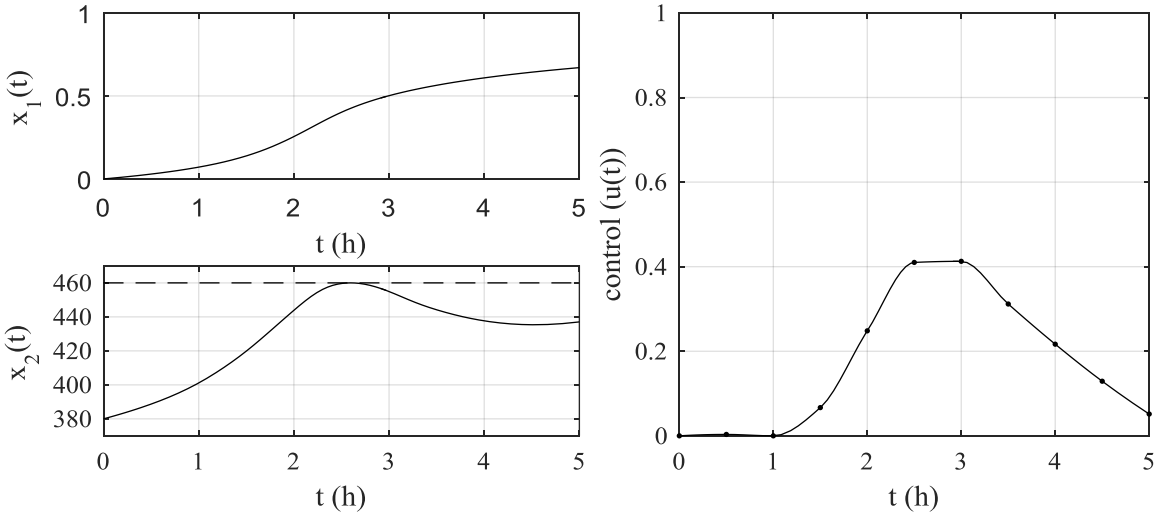


Figure 6.59. Optimal state and control trajectories of the Plug-Flow Reactor Temperature Control Problem via CVP-DCT for *Ndctc*=7 and *Ntgrid*=11.

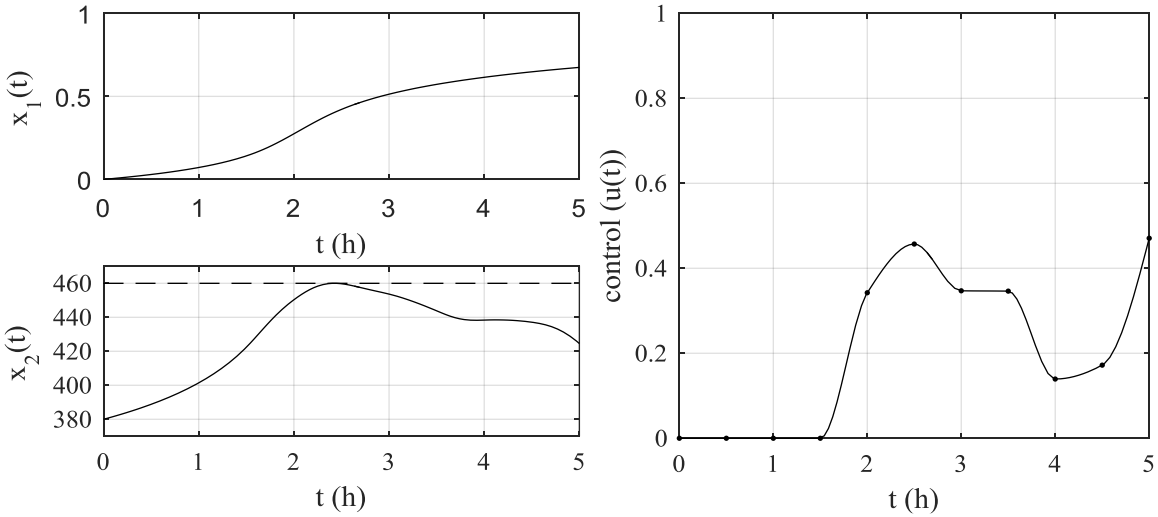


Figure 6.60. Optimal state and control trajectories of the Plug-Flow Reactor Temperature Control Problem via CVP-DCT for *Ndctc*=10 and *Ntgrid*=11.

It can be seen from the figures that the inequality path constraint, $x_2(t) < 460$, were not violated at any time in the interval $t \in [0,5]$.

6.5.4. Comparison of the Solutions of the Plug Flow Reactor Problem

The Temperature Control in a Plug Flow Reactor problem was solved by several authors. The reported solutions are compared in Table 6.23. The best performance index among the reported solutions (J*=0.6803) belongs to Reddy and Husain (1981). Among the methods used in this thesis work, CVO with *Ntgrid*=21 has the value closest to it. The solution obtained by the CVP-DCT method with *Ndctc*=4 and *Ntgrid*=11 has 2.27% relative difference and this solution requires only four decision variables. Among the performance indexes obtained by the CVP-DCT method, *Ndctc*=10 and *Ntgrid*=11 has the best performance index and it has 0.53% relative difference.

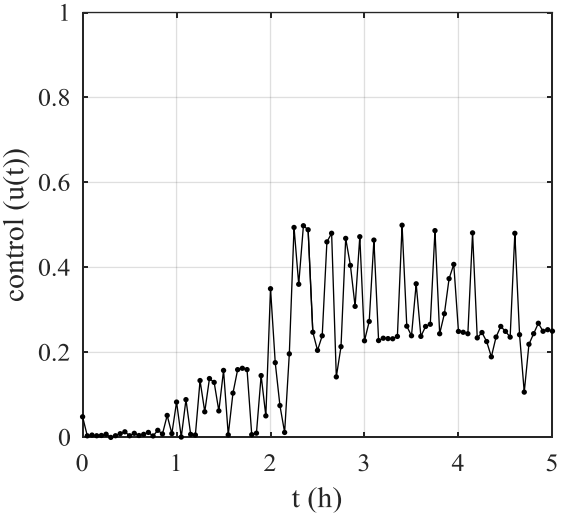
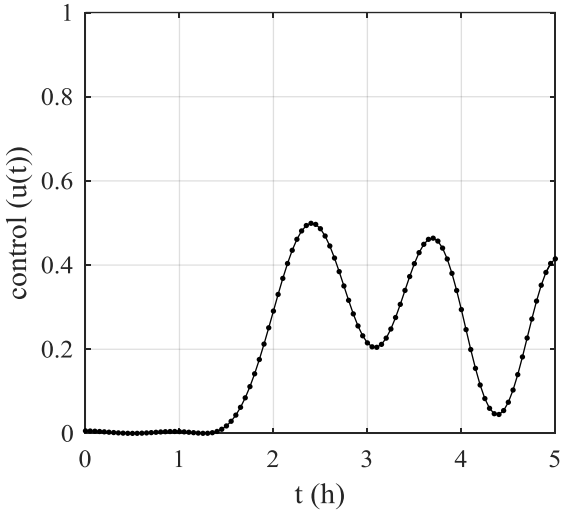
Table 6.23. Performance of different methods for the Plug Flow Reactor Problem.

Method	**J^***	**Relative difference (%)**
Conjugate Gradient Method (Reddy and Husain, 1981)	0.6803	-
Combined Mode Method (Ko and Steve, 1971)	0.68	0.04
Iterative Dynamic Programming (Mekarapiruk and Luus, 1997)	0.67726	0.45
Piecewise Constant CVP (Hirmajer *et al*., 2009)	0.67719	0.46
CVO (*Ntgrid* = 21)	0.67671	0.53
Iterative Dynamic Programming (Luus, 2000)	0.67532	0.73
CVO (*Ntgrid* = 11)	0.67502	0.78
CVP-DCT (*Ndctc*=10, *Ntgrid* = 11)	0.67461	0.84
CVP-DCT (*Ndctc*=7, *Ntgrid* = 11)	0.67136	1.31
CVP-DCT (*Ndctc*=4, *Ntgrid* = 11)	0.66487	2.27
CVP with 3rd order polynomial	0.65893	3.14

6.5.5. A Further Illustration of the Power of the CVP-DCT Method in Compressive Parameterization of the Control Trajectory for the Plug-Flow Reactor Temperature Control Problem

In this final section, we numerically compare the solutions obtained by CVO with *Ntgrid*=101 and CVP-DCT with *Ndctc*=10 and *Ntgrid*=101. The solution with CVO required more computational effort than the solution with CVP-DCT. Firstly, on the same time grids (*Ntgrid* = 101), CVP-DCT yielded a better performance index. Moreover, it used fewer (one fifth of the CVO) optimization decision variables. With the CVO method, the ODE integrator was called 5.2 times more than it was called by the CVP-DCT method. Finally, the required CPU time was 4.7 times longer in the CVO method. Comparison of the computational statistics along with the corresponding control trajectories are given in Table 6.24.

Table 6.24. Comparison of CVO and CVP-DCT via Computational Statistics for the Plug Flow Reactor Temperature Control Problem.

CVO with *Ntgrid*=101	CVP-DCT with *Ntgrid*=101, *Ndctc*=10
	
Number of decision variables : 101	Number of decision variables : 10
J* = 0.66110	J* = 0.67215
"fmincon" stats 8 iterations 1815 objective function evaluations	"fmincon" stats 7 iterations 277 objective function evaluations
DE stats 371 iterations 7420 objective function evaluations	DE stats 296 iterations 5920 objective function evaluations
Number of ODE function evaluations: 10,030,968 CPU: 1488 seconds	Number of ODE function evaluations: 1,894,178 CPU: 318 seconds

6.6.The Jacobson and Lele Problem

The problem was proposed by Jacobson and Lele (1969). The optimal-control problem consists of two states. Performance index, J, is an integral functional and we defined it as an axiullary ODE with an initial guess of zero. Therefore the system model with the

initial conditions are given as follows, where the third ODE's right-hand side is the integrand of the objective functional in the original formulation given in terms of two states.

$$\dot{x}_1 = x_2 \tag{6.41}$$

$$\dot{x}_2 = -x_2 + u \tag{6.42}$$

$$\dot{x}_3 = x_1^2 + x_2^2 + 0.005\, u^2 \tag{6.43}$$

$$\mathbf{x}(0) = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \tag{6.44}$$

The state path constraint is given as:

$$x_2(t) - 8(t - 0.5)^2 + 0.5 \leq 0 \tag{6.45}$$

To convert this constrained problem to an unconstrained optimization problem, the following PCP function was used.

$$PC = x_2(t_j) - 8(t_j - 0.5)^2 + 0.5 \tag{6.46}$$

$$PCP = \sum_{j=1}^{Ntgrid} \mu_{pc}\, max\{0,\ PC(t_j)\} \tag{6.47}$$

The penalty parameter (μ_{pc}) was taken as 50. The performance index is to minimize x_3 at the final time t_f which is equal to 1 hr. The augmented performance index became:

$$J = x_3(t_f) + PCP \tag{6.48}$$

6.6.1. Solution of Jacobson and Lele Problem with the CVO Method

The problem was solved with *Ntgrid*=11 and *Ntgrid*=21, and the calculated J^* values are 0.17026 and 0.17291, respectively. Solution with *Ntgrid*=21 reached the maximum iteration limit, therefore the optimization stopped prematurely. The initial guess was taken as $u(t_k) = 0$, k=1, ..., *Ntgrid*. The interpolation method was selected as pchip. The optimal

state and control trajectories and the trajectory of path constraint obtained by *Ntgrid*=11 are given on Figures 6.61.

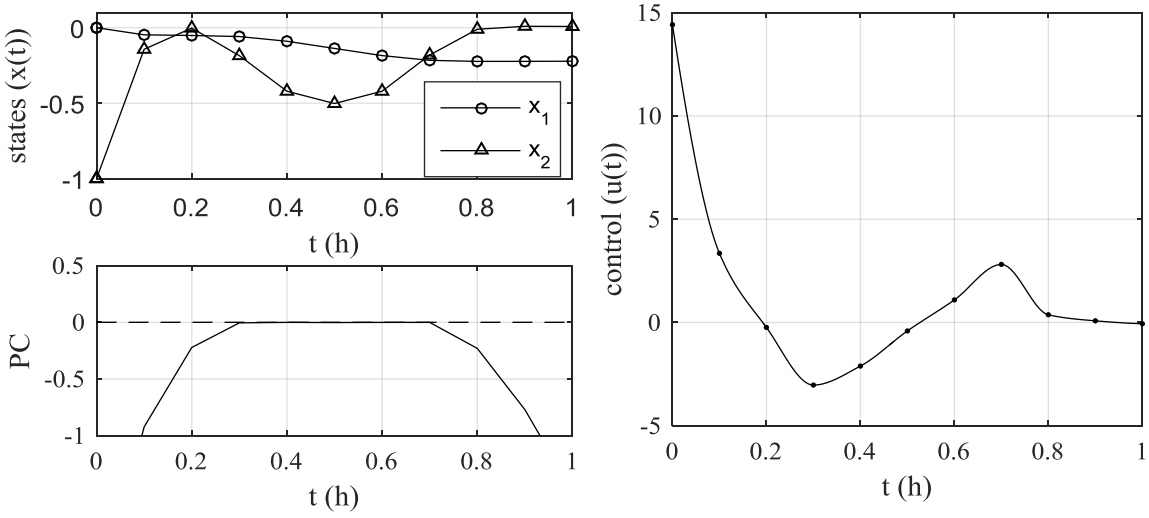


Figure 6.61. Optimal state and control trajectories of the Jacobson and Lele Problem via CVO for *Ntgrid* =11.

6.6.2. Solution of the Jacobson and Lele Problem with the CVP Method

This problem was also solved by the CVP method with seven functional forms that had been discussed in Chapter 4. Among the functions used, only the the 5^{th}-order polynomial yielded a solution which was similar to the reported solutions in the literature. The initial guesses of all the parameters were taken as zero. The minimized performance index was 0.18675 and the corresponding optimal state and control trajectories and path constraint trajectory are given in Figure 6.62.

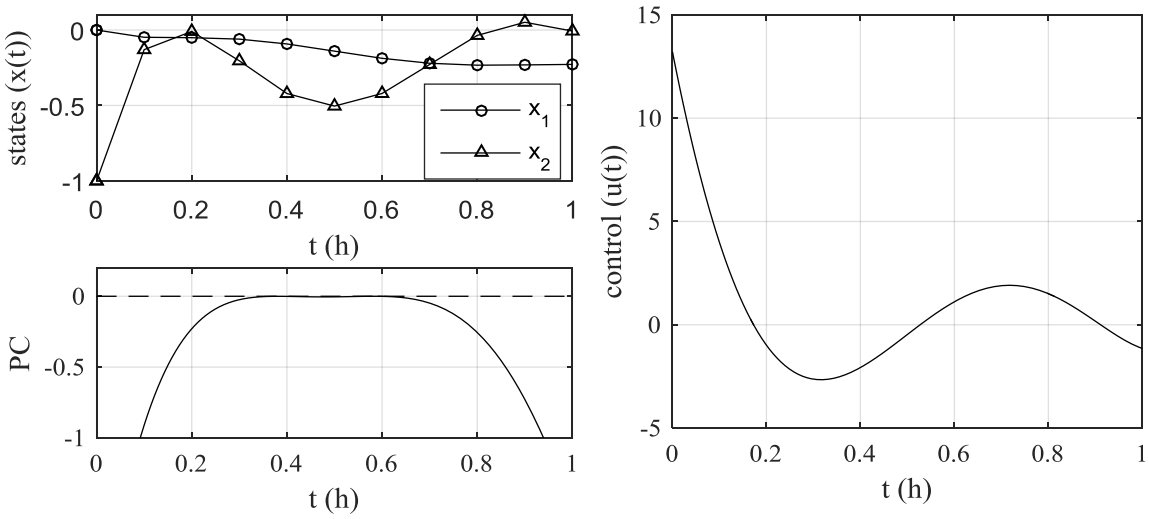


Figure 6.62. Optimal state and control trajectories of the Jacobson and Lele Problem via CVP with 5th-order polynomial.

6.6.3. Solution of the Jacobson and Lele Problem with the CVP-DCT Method

The problem was solved with *Ntgrid* = 11 by using different *Ndctc* values. The initial guess was taken as DCT of constant trajectory of $u(t_k) = 0$, k=1, ..., *Ntgrid*. The upper and lower bounds on the decision variables (DCTCs) were set as −1000 and +1000 for "fmincon". The optimal performance indexes are given in Table 6.25.

Table 6.25. Optimal performance indexes of the Jacobson and Lele Problem by CVP-DCT on 11 time grids for different *Ndctc*.

Ndctc	$\mathbf{J}^*$	***Ndctc***	$\mathbf{J}^*$	***Ndctc***	$\mathbf{J}^*$
2	0.46776	5	0.18137	8	0.17377
3	0.34716	6	0.17611	9	0.17311
4	0.19373	7	0.17573	10	0.17063

For this example, optimum performance indexes were very close to each other when *Ndctc* ≥ 6. The relative difference between the optimal performance indexes obtained with *Ndctc*=6 and *Ndctc*=9 is 1.73%, the difference between the solutions with *Ndctc*=9 and

Ndctc=10 is 1.45%. The optimum state and control trajectories and the path constraint trajectory obtained with *Ndctc* = 6 and *Ndctc* = 10 are given by Figure 6.63 and 6.64.

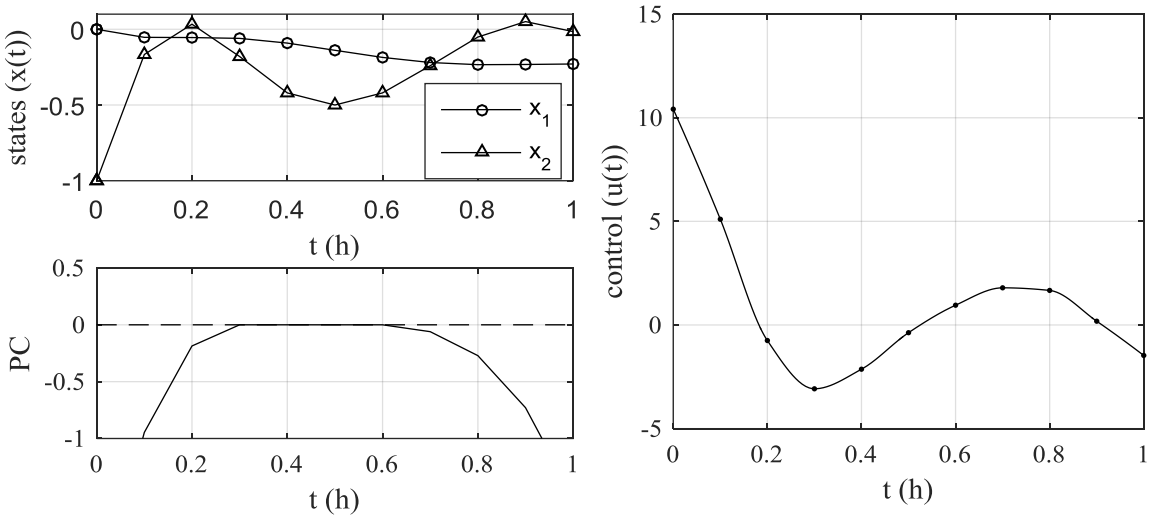


Figure 6.63. Optimal state and control trajectories of the Jacobson and Lele Problem via CVP-DCT for *Ndctc*=6 and *Ntgrid*=11.

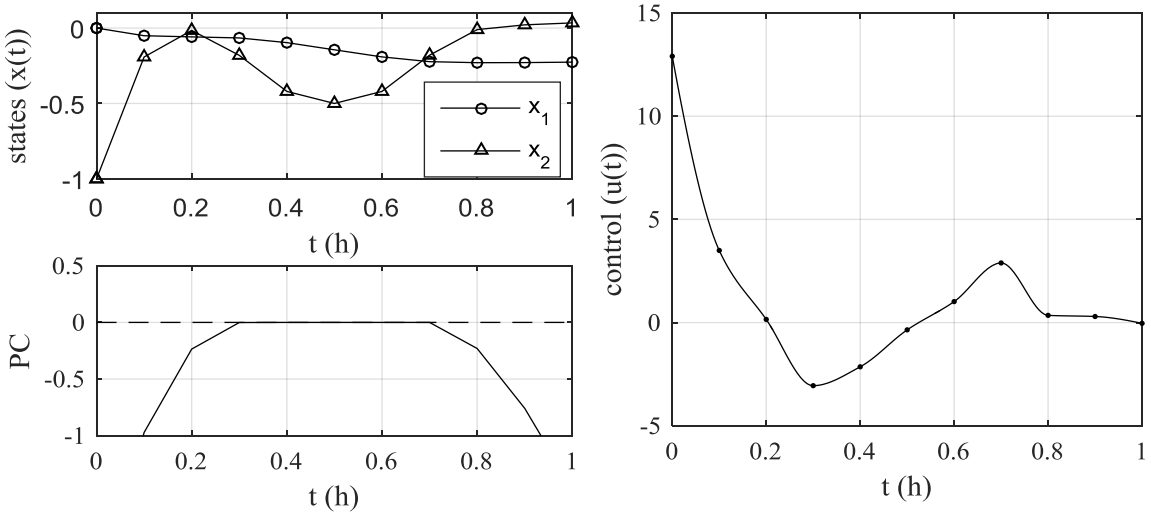


Figure 6.64. Optimal state and control trajectories of the Jacobson and Lele Problem via CVP-DCT for *Ndctc*=10 and *Ntgrid*=11.

6.6.4. Comparison of the Solutions of the Jacobson and Lele Problem

As discussed in Chapter 4, handling path constraints is difficult in sequential method. This problem required high CPU times even with few decision variables. Path constraint violations smaller than 10^{-4} were neglected. This problem was solved by several authors. The reported solutions are given in Table 6.26. Among the reported solutions, the best performance index was reported as 0.16946 by Neuman and Sen (1973). CVO with *Ntgrid*=11 found a solution with 0.47% relative difference. The solutions obtained by the CVP-DCT method with *Ndctc*=6 had 3.92% relative difference and this difference decreased to 0.69% when *Ndctc*=10 was taken.

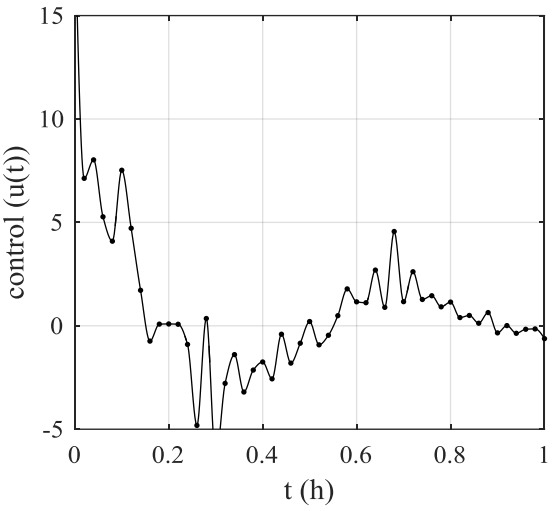
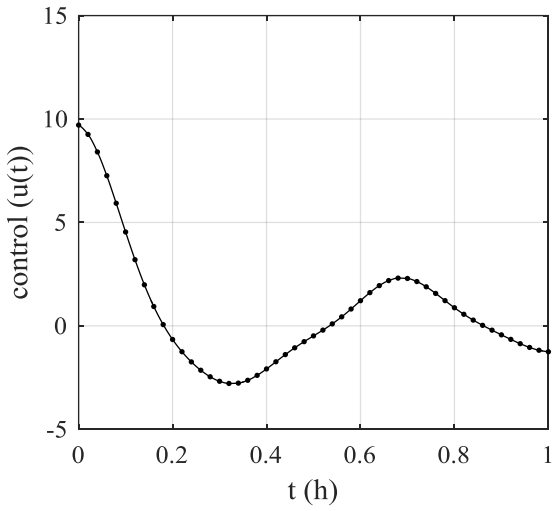
Table 6.26. Performance of different methods for the Jacobson and Lele Problem.

Method	**J***	**Relative difference (%)**
Collocation(Neuman and Sen, 1973)	0.16946	-
Direct Collocation (Logsdon and Biegler, 1989)	0.1696	0.08
Pseudospectral Collocation (PROPT)(Rutquist *et al*., 2010)	0.16982	0.21
Control Parametrization (Feehery, 1998)	0.16983	0.22
Orthogonal Collocation on Finite Elements (DYNOPT) (Cizniar, *et al*., 2005)	0.17014	0.40
CVO (*Ntgrid*=11)	0.17026	0.47
CVP-DCT (*Ndctc*=10, *Ntgrid*=11)	0.17063	0.69
Piecewise Constant CVP (Hirmajer *et al*., 2009)	0.17264	1.88
CVP-DCT (*Ndctc*=9, *Ntgrid*=11)	0.17311	2.15
CVP-DCT (*Ndctc*=6, *Ntgrid*=11)	0.17611	3.92

6.6.5. A Further Illustration of the Power of the CVP-DCT Method in Compressive Parameterization of the Control Trajectory for the Jacobson and Lele Problem

In this final section, we numerically compare the solutions obtained by CVO with *Ntgrid*=51 and CVP-DCT with *Ndctc* = 10 and *Ntgrid* = 101. The solution with CVO by taking *Ntgrid*=101 required extensive computational effort and it could not provide a stable solution. Therefore, the problem was solved with *Ntgrid*=51 and the maximum iteration limits of the DE algorithm was taken as 3500 to get a converged solution. Firstly, on the same time grids (*Ntgrid* = 51), CVP-DCT yielded a better performance index. Moreover, it used fewer (one fifth of the CVO) optimization decision variables. Comparison of the optimization statistics of both methods showed that "fmincon" required more function evaluations for CVO and this is mostly caused by the large dimension of the Hessian matrix. It should be noted that the DE solution reported here corresponds to the value obtained at the maximum iteration limit of 3500. With the CVO method, the ODE integrator was called 20 times more than it was called in the CVP-DCT method. Finally, the required CPU time was 18 times longer in the CVO method. The comparison of computational statistics along with the corresponding control trajectories are given in Table. 6.27.

Table 6.27. Comparison of CVO and CVP-DCT via Computational Statistics for the Jacobson and Lele problem.

CVO with *Ntgrid*=51	CVP-DCT with *Ntgrid*=51, *Ndctc*=10
	
Number of decision variables : 51	Number of decision variables : 10
J* = 0.17999	J* = 0.17440
<u>"fmincon" stats</u> 49 iterations 6231 objective function evaluations	<u>"fmincon" stats</u> 16 iterations 688 objective function evaluations
<u>DE stats</u> 3500 iterations 70000 objective function evaluations	<u>DE stats</u> 527 iterations 10540 objective function evaluations
Number of ODE function evaluations: 104,027,095	Number of ODE function evaluations: 5,714,2014
CPU: 9501 seconds	CPU: 548 seconds

6.7.The Two Stage CSTR Problem

The Two Stage CSTR problem consists of two-stage nonlinear CSTR system in which a first order irreversible reaction, $A \rightarrow B$, takes place in each reactor. This system model is based on original CSTR model of Aris and Amundson (1958) and the optimal problem formulation is taken from Dadebo and Luus (1992).

$$\dot{x}_1 = 0.5 - x_1(t) - R_1 \tag{6.49}$$

$$\dot{x}_2 = -2\,(x_2(t) + 0.25) - u_1(t)(x_2(t) + 0.25) + R_1 \tag{6.50}$$

$$\dot{x}_3 = x_1(1 - \tau) - x_3(t) - R_2 + 0.25 \tag{6.51}$$

$$\dot{x}_4 = x_2(1 - \tau) - 2x_4(t) - u_2(x_4(t) + 0.25) + R_2 - 0.25 \tag{6.52}$$

where τ is the time delay and taken as 0.1. $x_1(t)$ and $x_3(t)$ represent the scaled concentration of Tank 1 and Tank 2 while $x_2(t)$ and $x_4(t)$ represent the scaled temperature of Tank 1 and Tank 2. The temperature control inputs are represented as $u_1(t)$ for Tank 1 and $u_2(t)$ for Tank 2. The reaction terms R_1 and R_2 are defined as follows.

$$R_1 = (x_1(t) + 0.5)\, e^{25x_2(t)/(x_2(t)+2)} \tag{6.53}$$

$$R_2 = (x_3(t) + 0.5)\, e^{25x_4(t)/(x_4(t)+2)} \tag{6.54}$$

The performance index, J, is to minimize the function in Lagrange form and the final time is given as $t_f = 2$.

$$J = \int_0^{tf} (x_1^2 + x_2^2 + x_3^2 + x_4^2 + 0.1\,u_1^2 + 0.1\,u_2^2)dt \tag{6.55}$$

The performance index is handled as an auxilarry state variable and written in Mayer form as follows.

$$\dot{x}_5 = x_1^2 + x_2^2 + x_3^2 + x_4^2 + 0.1u_1^2 + 0.1u_2^2 \tag{6.56}$$

The initial conditions are given as follows.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0.15 \\ -0.03 \\ 0.1 \\ 0 \\ 0 \end{bmatrix} \tag{6.57}$$

The normalized control inputs have upper and lower bound constraints.

$$-1 \leq u_1 \leq 1 \tag{6.58}$$

$$-1 \leq u_2 \leq 1 \tag{6.59}$$

The constrains were handled by The Upper-Lower Penalty Function (ULP) as given in Equation (4.8). Finally, the objective function to minimize became as follows.

$$\mathrm{J} = \mathrm{x}_5 + \mathrm{ULP} \tag{6.60}$$

6.7.1. Solution of the Two Stage CSTR Problem with the CVO Method

The problem was solved with *Ntgrid*=11. Since this problem has two control inputs, the number of decision variables is equal to twice as *Ntgrid*, i.e. optimization algorithm used 22 decision variables. The calculated J^* value is 0.023240. The initial guesses of both control inputs were taken as $u_i(t_k) = 0$, k=1, ..., *Ntgrid* and i=1,2. The interpolation method was selected as pchip. The optimal state and control trajectories are given in Figure 6.65.

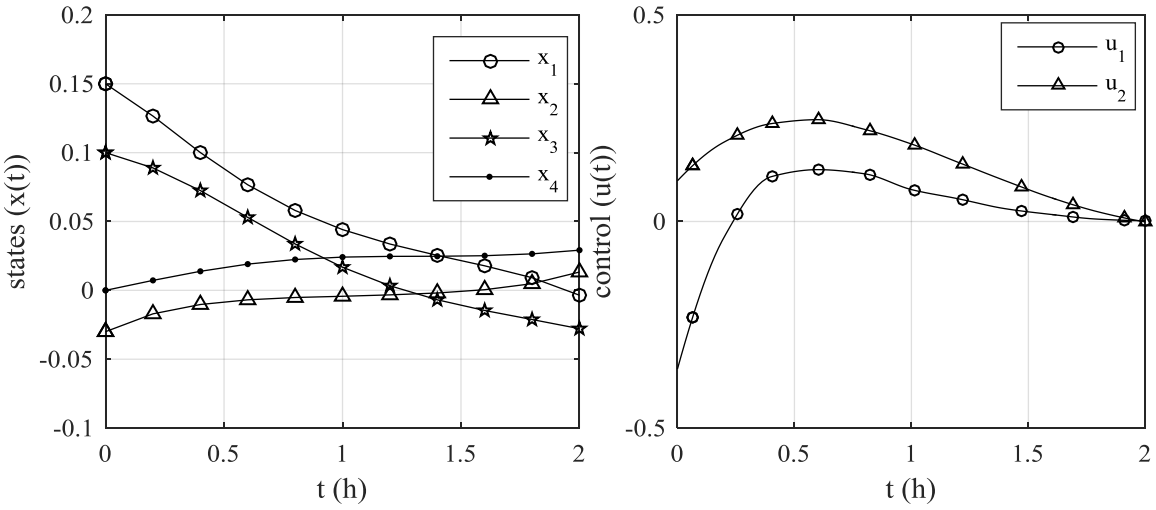


Figure 6.65. The optimal state and control trajectories of the Two Stage CSTR Problem via CVO *Ntgrid*=11.

6.7.2. Solution of the Two Stage CSTR Problem with the CVP Method

This problem was also solved by the CVP method with the seven functional forms that had been discussed in Chapter 4. The initial guesses of the parameters were taken as zero. The optimum performance indexes obtained are tabulated in Table 6.28. Functional form (vi) worked well and yielded the best performance indexes among other forms. Functional form (vii) used 12 decision variables whereas functional form (vi) used 10 decision variables but yielded a slightly better solution. The optimum state and control trajectories of the solutions are given in Figure 6.66 and 6.67. It should be noticed that the markers on the control trajectories do not represent grid values; their purpose is to provide better discrimination between the two curves.

Table 6.28. Functional forms used in the CVP to solve Two Stage CSTR Problem and the corresponding J* values.

Functional form	**J***
(i) Constant + exponential function	0.024569
(ii) Constant + exponential term	0.025865
(iii) Constant + 1st order polynomial + exponential term	0.026494
(iv) 2nd order polynomial	0.024145
(v) 3rd order polynomial	0.023367
(vi) 4th order polynomial	0.023246
(vii) 5th order polynomial	0.023288

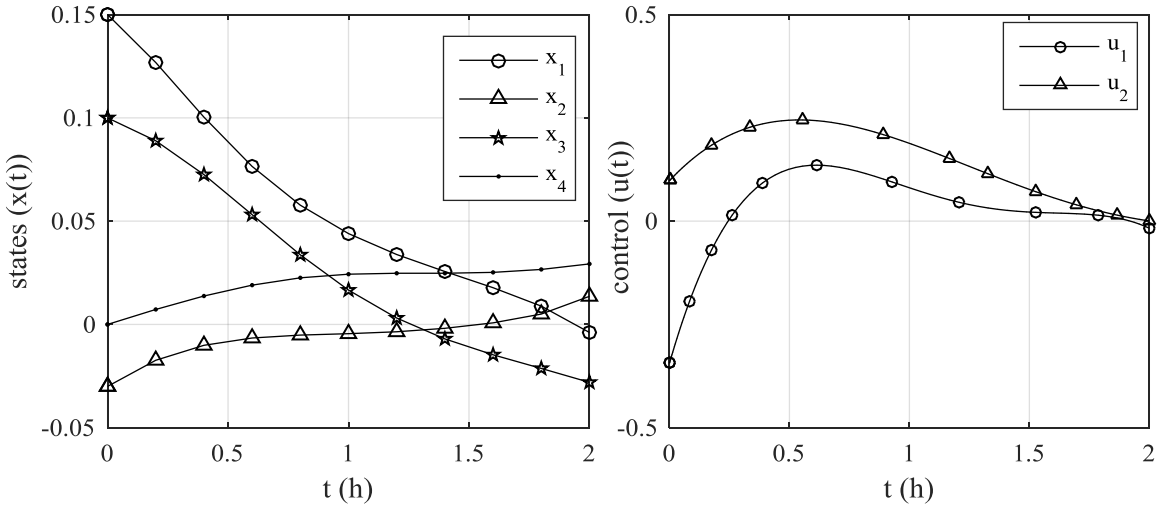


Figure 6.66. The optimal state and control trajectories of the Two Stage CSTR Problem with the CVP method with functional form (vi).

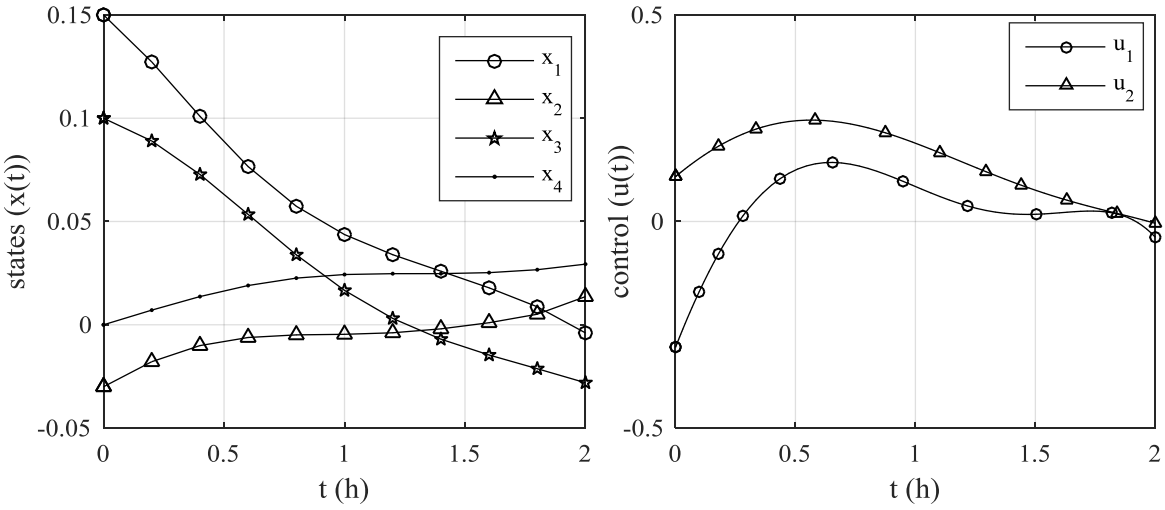


Figure 6.67. The optimal state and control trajectories of the Two Stage CSTR Problem with the CVP method with functional form (vii).

6.7.3. Solution of the Two Stage CSTR Problem with the CVP-DCT Method

The problem was solved with *Ntgrid*=11 by using different *Ndctc* values. The initial guess was taken as DCT of constant trajectory of $u(t_k) = 0$, k=1, ..., *Ntgrid*. The upper and lower bounds on the decision variables (DCTCs) were set as -1 and +1 for "fmincon". The optimal performance indexes are given in Table 6.29.

Table 6.29. Optimal performance indexes of the Two Stage CSTR Problem by CVP-DCT on 11 time grids for different *Ndctc*.

Ndctc	$\mathbf{J}^*$	***Ndctc***	$\mathbf{J}^*$	***Ndctc***	$\mathbf{J}^*$
2	0.026648	5	0.023444	8	0.023268
3	0.024377	6	0.023332	9	0.023249
4	0.023791	7	0.023272	10	0.023240

Due to the form of the control trajectories of this problem, with the same number of decision variables the CVP-DCT method found slightly poorer performance indexes then the CVP method. If *Ndctc* ≥ 5 is used, the relative differences of the optimum performance indexes obtained by the consecutive *Ndctc* values are less than 0.5% and these differences

decrease with increasing *Ndctc*. It should be noted that the relative difference between *Ndctc*=5 and *Ndctc*=10 is only 0.88%. The optimum state and control trajectories by *Ndctc*=5, *Ndctc*=7, and *Ndctc*=10 are given by Figure 6.68, 6.69 and 6.70.

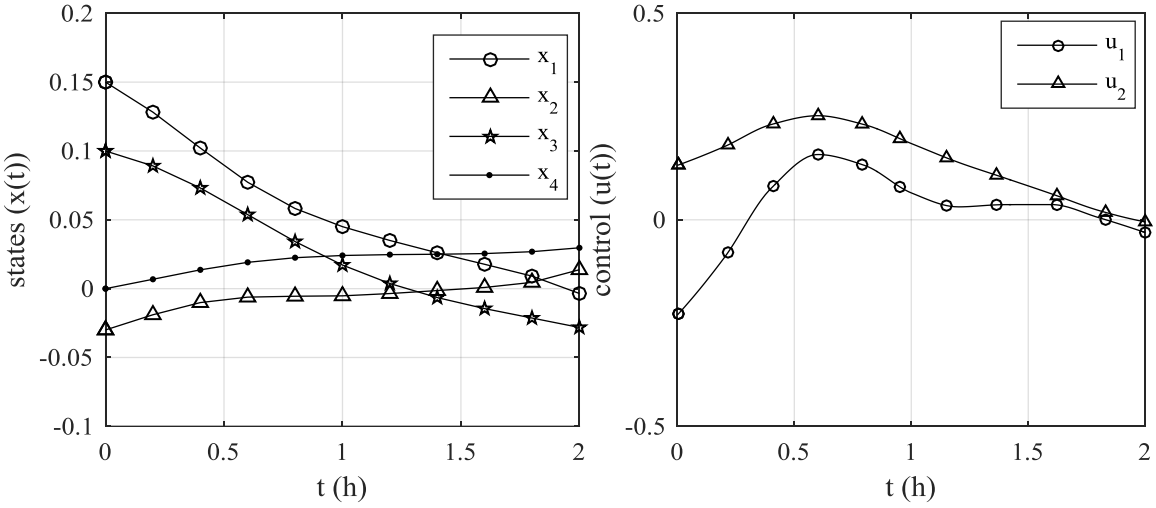


Figure 6.68. Optimal state and control trajectories of the Two Stage CSTR Problem via CVP-DCT for *Ndctc*=5 and *Ntgrid*=11..

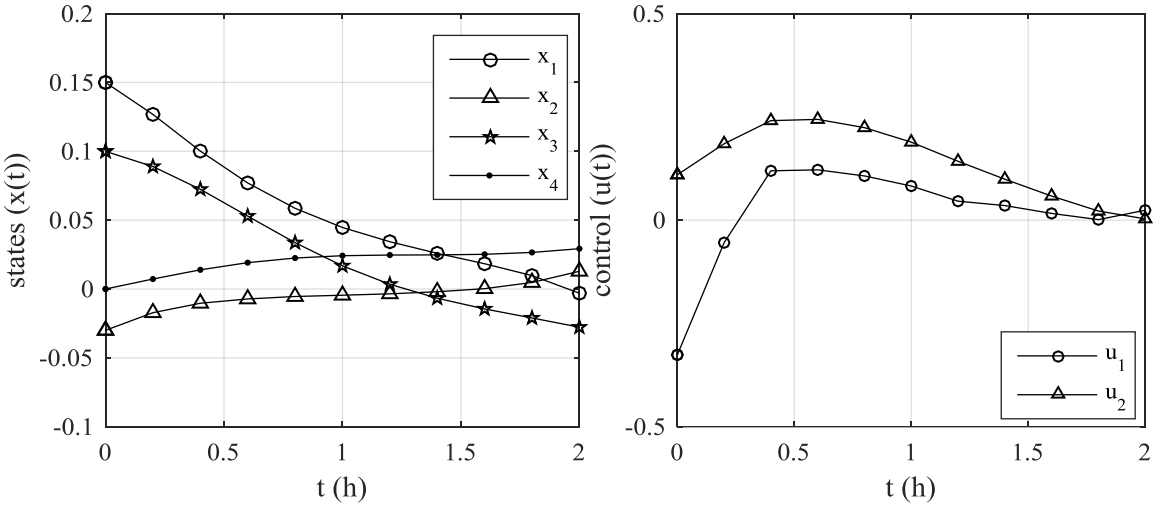


Figure 6.69. Optimal state and control trajectories of the Two Stage CSTR Problem via CVP-DCT for *Ndctc*=8 and *Ntgrid*=11.

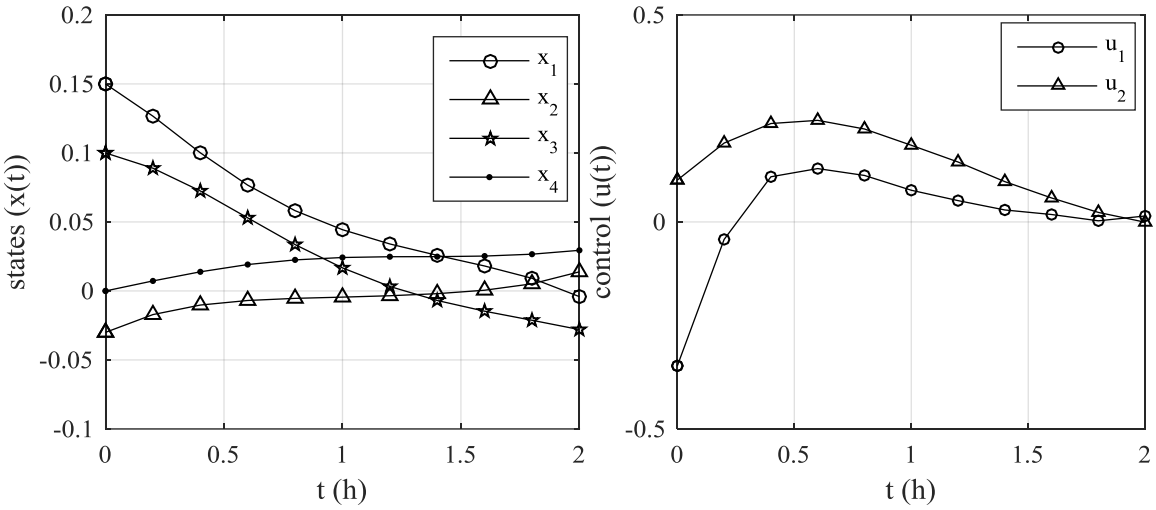


Figure 6.70 Optimal state and control trajectories of the Two Stage CSTR Problem via CVP-DCT for *Ndctc*=10 and *Ntgrid*=11.

6.7.4. Comparison of The Solutions of the Two Stage CSTR Problem

The problem was solved by several authors in the literature. The best performance index among the reported solutions (J*=0.23166) belongs to Luus (2000) in which the Iterative Dynamic Programming Method with Piecewise Linear Control approach is utilized. The reported solutions are compared in Table 6.30. Among the methods used in this thesis work, CVO with *Ntgrid*=11 (with 22 decision variables) had the value closest to it with a 0.32% relative difference. The solution obtained by the CVP-DCT method with *Ndctc*=4 had 2.69% relative difference and it should be noted that the number of optimization decision variables was eight in that case. Furthermore, the performance indexes obtained with the same number of parameters by CVP and CVP-DCT were also compared. For this example, with same number of decision variables, CVP found better performance indexes than CVP-DCT (e.g. compare the solutions which used 12 parameters by CVP-DCT with *Ndctc*=6 and by CVP with functional form (vi)). The CVP had a better performance than CVP-DCT's performance, because the control trajectory was more suitable for approximation within the selected functional forms of CVP.

Table 6.30. Performance of different methods for the Two Stage CSTR Problem.

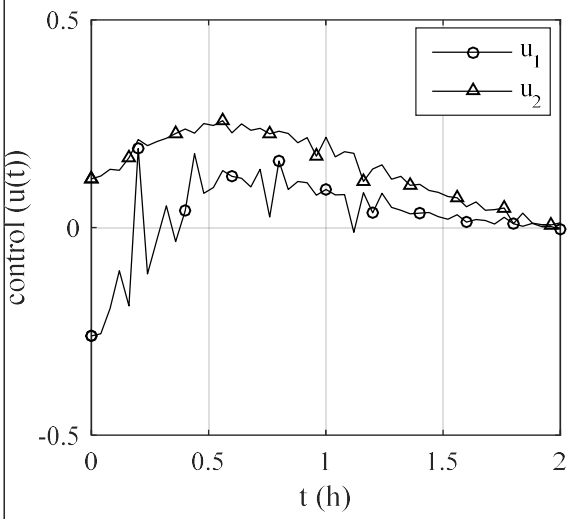
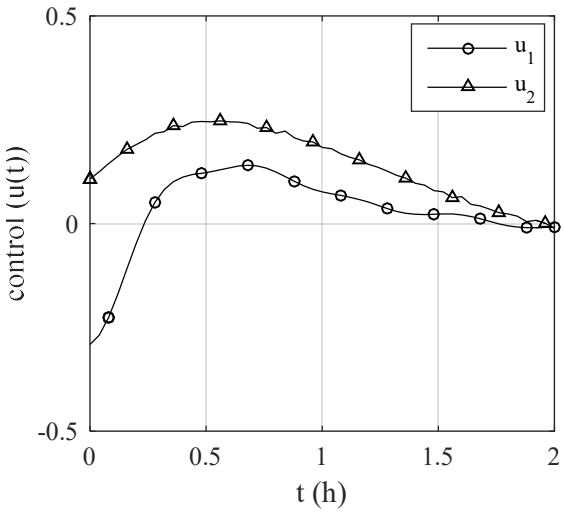
Method	**J***	**Relative difference (%)**
Iterative Dynamic Programming (with piecewise linear control) Luus (2000)	0.023166	-
Pseudospectral Collocation (PROPT) (Rutquist Edvall, 2010)	0.023238	0.3108
CVO (with *Ntgrid*=11)	0.023239	0.3151
CVP-DCT (*Ndctc*=10, *Ntgrid* = 11)	0.023240	0.3194
CVP with functional form (vi)	0.023250	0.3453
CVP-DCT (*Ndctc*=9, *Ntgrid* = 11)	0.023249	0.3583
CVP-DCT (*Ndctc*=8, *Ntgrid* = 11)	0.023268	0.4403
Iterative Dynamic Programming (with piecewise constant control) Luus (2000)	0.023270	0.4489
CVP-DCT (*Ndctc*=7, *Ntgrid* = 11)	0.023272	0.4576
Control Vector Iteration (Oh and Luus, 1977)	0.02328	0.4921
CVP with functional form (vii)	0.02329	0.5266
Direct Search Method(Oh and Luus, 1977)	0.02330	0.5784
CVP-DCT (*Ndctc*=6, *Ntgrid* = 11)	0.02333	0.7166
CVP with functional form (v)	0.02337	0.8677
CVP-DCT (*Ndctc*=5, *Ntgrid* = 11)	0.02344	1.2000
CVP-DCT (*Ndctc*=4, *Ntgrid* = 11)	0.02379	2.6979

6.7.5. A Further Illustration of the Power of the CVP-DCT Method in Compressive Parameterization of the Control Trajectory for the Two Stage CSTR Problem

In this final section, we numerically compare the solutions obtained by CVO with *Ntgrid*=51 and CVP-DCT with *Ndctc*=10 and *Ntgrid*=51. The solution with CVO required extensive computational effort and could not provide a stable solution. Therefore, the problem was solved and the maximum iteration limits of the DE algorithm was taken as

3000 to get a converged solution. Firstly, on the same time grids (*Ntgrid*=51), CVP-DCT yielded a better performance index. Moreover, it used fewer (one fifth of the CVO) optimization decision variables. In the solution by CVO, "fmincon" found a local solution (J=0.08) therefore it stopped after 4^{th} iteration. On the other hand, the local minimum which "fmincon" found with CVP-DCT was close to the reported solution. It should be noted that the DE solution reported here corresponds to the value obtained at the maximum iteration limit of 3000. With the CVO method, the ODE integrator was called 8.4 times more than it was called in the CVP-DCT method. Finally, the required CPU time was 8.3 times longer in the CVO method. The comparison of computational statistics along with the corresponding control trajectories are given in Table 6.31.

Table 6.31. Comparison of CVO and CVP-DCT via Computational Statistics for the Two Stage CSTR Problem.

CVO with *Ntgrid*=51	CVP-DCT with *Ntgrid*=51, *Ndctc*=10
	
Number of decision variables : 102	Number of decision variables : 20
J* = 0.023616	J* = 0.023262
<u>"fmincon" stats</u> 4 iterations 915 objective function evaluations	<u>"fmincon" stats</u> 18 iterations 1151 objective function evaluations
<u>DE stats</u> 3000 iterations 60000 objective function evaluations	<u>DE stats</u> 1070 iterations 21400 objective function evaluations
Number of ODE function evaluations: 65,000,027 CPU: 8055 seconds	Number of ODE function evaluations: 7,697,081 CPU: 974 seconds

7. CONCLUSIONS AND FUTURE RECOMMENDATIONS

This thesis focused on the numerical solution methods to for open-loop optimal-control problems. Among the numerical solution methods that appear in the literature, the "direct sequential approach" was adopted, and a novel method, Compressive Control-Vector Parameterization with Discrete Cosine Transform (CVP-DCT) was proposed and developed. The aim was to eliminate some of the disadvantages of other sequential methods.

Several optimal-control problems were solved in Chapter 6. Since the direct methods often lead to local solutions, an optimization sequence consisting of a gradient-based algorithm followed by a population-based algorithm was selected. In all the example problems, the CVP-DCT method firstly found a local solution via the gradient-based optimizer then that solution was improved by the population-based optimizer. In Problem 1 given in Section 6.1, the global (analytical) solution was used to compare it with the numerical solutions. The comparison showed that the CVP method, with the same number of optimization decision variables, yielded slightly better solutions compared to the CVP-DCT method because the functional form of the optimal control trajectory was very similar to the trajectories that can be represented by the proposed functional forms for the CVP method. However, in the second problem in Section 6.2 (The Catalyst Mixing Problem), the CVP method could not yield a solution better than the solution of the CVP-DCT method, even with a high number of parameters. In Section 6.3 (The Van der Pol Oscillator Problem) and Section 6.4 (The Consecutive Batch Reaction Problem), all the CVP, CVO, and CVP-DCT methods worked well and yielded solutions that were close to each other. However, in the problems with state path constraints given in Section 6.5 (Plug Flow Reactor Problem) and in Section 6.6 (Jacobson and Lele Problem), due to the complex form of the control trajectories, the CVP method failed, yet the CVP-DCT method worked well. In Section 6.7 (Two Stage CSTR), an example problem with two-dimensional controls was given. With this example, it was demonstrated that for optimal-control problems with multiple controls, the proposed CVP-DCT method was indeed very flexible and practical owing to the possibility of using the DCT independently for each control input.

In conclusion, if the optimal control trajectory has a known or forecastable profile and if this profile can be represented by a simple function of time with only few parameters, then the CVP method can be the best choice. However, the CVP-DCT method is independent of the functional form. In other words, it does not require a priori knowledge of the control trajectory. Therefore, the CVP-DCT method is more flexible to apply than the CVP method. In the final section of each problem, a more detailed comparison of the CVO and the CVP-DCT methods, in terms of computational measures, were provided. In all cases, it was confirmed that, for large numbers of time grids, the CVP-DCT method could break the curse of dimensionality associated with the CVO method.

The advantages of the proposed CVP-DCT method can be summarized as follows.

- The CVP-DCT method does not require a priori knowledge of the shape and complexity of the control trajectory. Therefore, it can be used in any optimal control problem without prespecification.
- The CVP-DCT method, with only a few (two to four) parameters, can provide a very good (suboptimal) initial guess trajectory to other more sophisticated but initial-guess sensitive numerical solution methods.
- If the optimum control trajectory is a simple and smooth, the CVP-DCT method can provide solutions which are very close to the global solution using only a few decision variables.
- The CVP-DCT method requires much less CPU time compared to CVO method with the same number of time grids.
- When more DCT coefficients, i.e., higher number of decision variables, are used, the CVP-DCT method has the potential to generate (reconstruct) more complex control trajectories (see Chapter 5).
- The performance (required number of DCT coefficients, number of optimization decision variables) of the CVP-DCT method is independent of the dimension of the state equations (number of ODEs).
- The performance (required number of DCT coefficients, number of optimization decision variables) of the CVP-DCT method is independent of the dimension of the time grid. Even two decision variables (two DCT coefficients) are enough to

reconstruct (can unfold to) control vector values over e.g., 1000 or million grids. The CPU time, thus, basically is independent of the total number of time grids.

The disadvantages of the proposed CVP-DCT method can be summarized as follows.

- The use of more DCT coefficients, i.e., higher numbers of decision variables, upon reconstruction (due to cosine series inherent in the DCT formula), tends to lead to more fluctuant control trajectories, and this adversely affects the optimization task. These artifacts in the reconstructed control trajectories make the assessment of the optimization results a more complicated task since it may be unclear whether these artifacts are due to cosine terms inherent in the DCT formula (i.e., natural) or due to premature termination of the optimizer or the failure of the optimizer to reach a good and smooth (perhaps the global) solution.
- For state models with no explicit time dependence (i.e., if the right-hand-sides of the ODEs do not contain "t" term explicitly), the total time derivative of the Hamiltonian must be constant at zero. As is the case in all "direct sequential" methods, the CVP-DCT method may not satisfy this condition on the total time derivative of the Hamiltonian. This is true especially when low number of DCT coefficients are used, i.e., when a very smooth and a very approximate yet very fast solution is aimed with the use of the CVP-DCT method.
- As is the case in all "direct sequential" methods which do not rely on the Hamiltonian function, the CVP-DCT method, as well, may not satisfy the necessary condition of optimality (i.e., vanishing partial derivatives of the Hamiltonian with respect to control variables at all time grids) or Pontryagin's Maximum Principle (Hamiltonian function must be at its minimum for all times). Therefore, as in many other "direct sequential" methods, the CVP-DCT method may not guarantee the global solution that is in accordance with the analytical solution (optimal-control theory).
- Unlike the CVO method, where the decision variables are nothing but identically the control-input values, in the CVP-DCT method, the decision variables are the DCT coefficients which are not directly related to the actual control-input values. In fact, without (before) the i-DCT, just based on the DCT coefficients, there is no way of estimating even the order-of-magnitude values of the control inputs. Therefore, for control-variable bound-constrained optimal-control problems where the upper and

lower bounds of the controls are specified, with the CVP-DCT method, these bound values cannot be directly given to the optimizer as simple bounds on the decision variables, but instead, they must be guessed regarding the DCT coefficients. Since making this guess is not a well-defined task and not easy, we suggest the use of an unconstrained optimization algorithm and the penalty function approach, as adopted in this thesis work.

- The CVP-DCT is a global parameterization approach, since the few early DCT coefficients are reconstructed via i-DCT to the entire time domain between the initial and final times, at once. For optimal control trajectories with very sharp changes or discontinuities (e.g., singular control problems), the CVP-DCT method cannot approximate the trajectory well enough with only few number of decision variables. If the number of decision variables (i.e., the number of DCT coefficients used) is increased to capture such discontinuities better, then the resulting trajectory may become oscillatory around the discontinuity. This behavior, however, is also encountered in many polynomial-approximation methods used in the numerical solution of optimal-control problems, e.g., in the case of expansion of a discontinuous trajectory with N^{th}-degree Chebyshev polynomials.

7.1.Recommendations for Future Studies

When the optimum control trajectory has sharp changes or discontinuities (e.g., singular control problems), the CVP-DCT method struggles to approximate the trajectory with few number of decision variables. If the number of DCT coefficients is increased to overcome this difficulty, the resulting trajectory may be fluctuant. In order to overcome such challenges and to improve the method, the CVP-DCT can be applied on finite elements (similar to the method of "orthogonal collocation on finite elements"). This can be implemented by dividing the time domain into a finite number of elements (stages) and apply the CVP-DCT method proposed in this thesis work to each element independently and by using only a small number of DCT coefficients (e.g., two to four) for each element. Each element's decision variables (DCT coefficients) can be decompressed to reconstruct the control trajectory in that element only. The continuity of the control trajectory itself and the

continuity of the slope (1st derivative) of the control trajectory across the elements (at the element boundaries) can easily be imposed using suitable penalty functions in the optimization formulation, or they can be given as the equality constraints if a constrained optimizer is used. This way, the proposed CVP-DCT method can create much more complex control trajectories including very abrupt changes or even discontinuities without using an excessive number of DCT coefficients (i.e., without using an excessive number of optimization decision variables).

Furthermore, the proposed CVP-DCT method can be utilized in "simultaneous direct methods" by parameterizing compressively both the state and control variables in terms of mutually exclusive DCT coefficients. This improvement may compensate the CPU effort that the sequential methods require mostly due to ODE integration and by (significantly) reducing the number of decision variables of the "simultaneous direct" methods.

Other data compression tools can be utilized to replace the DCT part of the proposed CVP-DCT method. One of such tools is the Discrete Wavelet Transform (DWT). If DCT is replaced with DWT then the "details coefficients" can be zeroed (neglected) and only the first few "approximation coefficients" of the wavelet decomposition can be used to compress the yet unknown control trajectories in terms of DWT "approximation coefficients". Such a method can be called as the CVP-DWT. Actually, CVP-DWT may perform better than CVP-DCT since the use of discontinuous mother wavelets, such as the Haar Wavelet, may furnish the envisioned CVP-DWT method with the inherent ability to cope with discontinuous control trajectories (e.g., in singular optimal-control problems).

Neural Networks (NNs) are known to have the ability to fit their outputs to any arbitrary function, however, at the expense of using more complex NN structures (more hidden layers and/or more neurons per hidden layer). The use of a NN to replace the DCT part of the proposed method will result in the CVP-NN method. However, this envisioned CVP-NN method is expected to require significantly more decision variables (more NN weights) and thus more CPU time compared to our CVP-DCT method. Furthermore, CVP-NN will have another uncertainty (burden) which is not present in our CVP-DCT method;

the task of deciding on the NN structure (number of layers, number of neurons in each layer, type of activation functions in each neuron, etc.). In fact, the use of NNs in parameterization of the optimal control trajectory is not a new idea, many examples and applications can be found in the literature. Indeed, due to such nuisances associated with NNs, the use of NNs in optimal-control problems is fading out in the literature due to the advancement of direct simultaneous methods. However, comparison of the CVP-NN method with the proposed CVP-DCT method may be an interesting future-work candidate.

Chebyshev Polynomials (CPs) have interesting properties and they are frequently used in variety of spectral-collocation methods for diverse applications. The word “spectral” comes from the fact that, for smooth functions (no discontinuities), the CPs converge fast, meaning that the function approximation power is compressed into the first few (early) CPs and the coefficients of the later CPs fade out quickly as the degree of polynomials increases. However, from signal and image processing literature it is known that the compressive power of CPs is less than that of the DCT (even though the DCT had been derived based on the CPs), and thus, basically, the CPs are not used in signal- and image-processing applications. However, comparison of an envisioned CVP-CP method with the proposed CVP-DCT method may be an interesting future-work candidate as well, especially for smooth control trajectories which both methods favor. As a final note, noticing that the word “spectral” stands for “compressive”, the proposed CVP-DCT method may also be termed as a “Spectral Control-Vector Parameterization with DCT”.

REFERENCES

Ahmed, N., Natarajan, T., Rao, K. R. (1974). "Discrete Cosine Transform", *IEEE Transactions on Computers*, *C-23*(1), 90–93.

Aris, R., Amundson, N. R. (1958). "An analysis of chemical reactor stability and control-I:The possibility of local control, with perfect or imperfect control mechanisms" *Chemical Engineering Science*, *7*(3), 121–131.

Balsa Canto, E., Banga, J. R., Alonso, A. A., Vassiliadis, V. S. (2002). "Restricted second order information for the solution of optimal control problems using control vector parameterization", *Journal of Process Control*, *12*(2), 243–255.

Banga, J. R., Balsa-Canto, E., Moles, C. G., Alonso, A. A. (2003). "Dynamic optimization of bioreactors—a review", In *Proceedings of the Indian National Science Academy. 69A* (pp. 257–265).

Banga, J. R., Irizarry-Rivera, R., & Seider, W. D. (1998). "Stochastic optimization for optimal and model-predictive control", *Computers & Chemical Engineering*, *22*(4–5), 603–612.

Bell, M. L., Sargent, R. W. H. (2000). "Optimal control of inequality constrained DAE systems", *Computers and Chemical Engineering*, *24*(11), 2385–2404.

Bellman, R. (1966). Dynamic Programming. *Science*, *153*(3731), 34–37.

Bertsekas, D. P. (1999). *Nonlinear programming*. *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability,* J. Neyman (Ed.), University of California Press, Berkeley, {CA}.

Betts, J. T. (2010). *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. *Control*, SIAM, Seattle

Biegler, L. T. (2007). "An overview of simultaneous strategies for dynamic optimization", *Chemical Engineering and Processing: Process Intensification*, *46*(11), 1043–1053.

Biegler, L. T. (2010). *Nonlinear Programming: concepts, algorithms and applications to*

chemical processes, SIAM, Philadelphia

Brusch, R. G., Schapelle, R. H. (1973). "Solution of Highly Constrained Optimal Control Problems Using Nonlinear Programing", *AIAA Journal*, *11*(2), 135–136.

Bryson, A. E. (1996). "Optimal control-1950 to 1985", *IEEE Control Systems Magazine*, *16*(3), 26–33.

Bryson, A. E., Ho, Y.-C. (1975). *Applied optimal control : optimization, estimation, and control*. Hemisphere Pub. Corp.

Canon, M. D., Clifton, D. C., Polak, E. (1969). *Theory of optimal control and mathematical programming,* McGraw-Hill.

Chakraborty, U. K. (2008). *Advances in differential evolution*. Springer Verlag.

Choose an ODE Solver MATLAB Version (2017a), https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html, accessed at June 2017.

Cizniar, M., Salhi, D., Fikar, M., Latifi, M. (2005). A MATLAB Package for Orthogonal Collocations on Finite Elements in Dynamic Optimisation. In *15 Int Conference Process Control* (pp. 58-1–7), Bratislava (Slovakia), June 7-10

Cuthrell, J. E., Biegler, L. T. (1987). "On the optimization of differential-algebraic process systems", *AIChE Journal*, *33*(8), 1257–1270.

Dadebo, S. A., Mcauley, K. B. (1995). "Dynamic optimization of constrained chemical engineering problems using dynamic programming", *Computers & Chemical Engineering*, *19*(5), 513–525.

Dadebo, S., Luus, R. (1992). "Optimal control of time-delay systems by dynamic programming", *Optimal Control Applications and Methods*, *13*(1), 29–41.

Feehery, W. F. (1998). *Dynamic Optimization with Path Constraints*, Ph.D. Thesis, Massachusetts Institute of Technology.

Fractional Brownian motion synthesis - MATLAB wfbm. https://www.mathworks.com/help/wavelet/ref/wfbm.html, accessed at June 2017.

Gämperle, R., Müller, S. D., Koumoutsakos, P. (2002). "A Parameter Study for Differential

Evolution", In *WSEAS International Conference. on Advances in Intelligent Systems* (pp. 293--298).

Goh, C. J., Teo, K. L. (1988). "Control parametrization: A unified approach to optimal control problems with general constraints", *Automatica*, *24*(1), 3–18.

Goldstine, H. H. (1980). *A History of the Calculus of Variations from the 17th through the 19th Century* (Vol. 5). New York, NY: Springer New York.

Gritsis, D. (1990). *The dynamic simulation and optimal control of systems described by index two differential-algebraic equations,* Ph.D. Thesis*,* Imperial College London (University of London).

Gunn, D. J., Thomas, W. J. (1965). "Mass transport and chemical reaction in multifunctional catalyst systems", *Chemical Engineering Science*, *20*(2), 89–100.

Hicks, G. A., Ray, W. H. (1971). "Approximation methods for optimal control synthesis", *The Canadian Journal of Chemical Engineering*, *49*(4), 522–528.

Hirmajer, T., Balsa-Canto, E., Banga, J. R. (2009). "DOTcvpSB, a software toolbox for dynamic optimization in systems biology", *BMC Bioinformatics*, *10*(1), 199.

Jacobson, D., Lele, M. (1969). "A transformation technique for optimal control problems with a state variable inequality constraint", *IEEE Transactions on Automatic Control*, *14*(5), 457–464.

Jain, A. K. (1989). *Fundamentals of digital image processing*. Prentice Hall.

Kirk, D. E. (2004). *Optimal Control Theory: An Introduction*. IEEE Transactions on Automatic Control.

Ko, D. Y. C., Steve, W. F. (1971). "Studies of singular solutions in dynamic optimization: I. Theoretical aspects and methods of solution", *AIChE Journal*, *17*(1), 249–252.

Kraft, D. (1985). "On Converting Optimal Control Problems into Nonlinear Programming Problems", In Schittkowski K. (Ed.), *Computational Mathematical Programming* (pp. 261–280). Springer Berlin Heidelberg.

Lewis, F. L., Vrabie, D. L., Syrmos, V. L. (2012). *Optimal Control.* John Wiley & Sons.

Liu, X., Chen, L., HU, Y. (2013). "Solution of Chemical Dynamic Optimization Using the

Simultaneous Strategies.", *Chinese Journal of Chemical Engineering*, *21*(1), 55–63.

Logsdon, J. S., Biegler, L. T. (1989). "Accurate solution of differential-algebraic optimization problems", *Industrial & Engineering Chemistry Research*, *28*(11), 1628–1639.

Luus, R. (2000). *Iterative dynamic programming*. *Journal of Chemical Information and Modeling* (Vol. 53).

Luus, R. (2001). "Boundary Condition Iteration", In C. Floudas (Ed.), *Encyclopedia of Optimization* (pp. 205–209). Boston, MA: Springer US.

Luus, R. (2008). "Control Vector Iteration", In C. Floudas (Ed.), *Encyclopedia of Optimization* (pp. 509–513). Boston, MA: Springer US.

Mallipeddi, R., Suganthan, P. N. (2010). *Differential Evolution Algorithm with Ensemble of Parameters and Mutation and Crossover Strategies*, Springer, Berlin, Heidelberg.

MATLAB version 2017a, Optimization Toolbox, https://www.mathworks.com/help/optim/ug/"fmincon".html, accessed at June 2017.

Mekarapiruk, W., Luus, R. (1997). "Optimal Control of Inequality State Constrained", Systems *Industrial & Engineering Chemistry Research*, 36(5), 1686-1694.

Morison, K. R. (1984). *Optimal control of processes described by systems of differential-algebraic equations,* Ph.D. Thesis. Imperial College London.

Morison, K. R., Sargent, R. W. H. (1986). Optimization of multistage processes described by differential-algebraic equations, *Numerical Analysis,* Springer, Berlin, Heidelberg.

Neuman, C. P., Sen, A. (1973). "A suboptimal control algorithm for constrained problems using cubic splines", *Automatica*, *9*(5), 601–613.

Oh, S. H., Luus, R. (1977). "Use of orthogonal collocation method in optimal control problems", *International Journal of Control*, *26*(5), 657–673.

Paulen, R., Fikar, M. (2016). *Optimal Operation of Batch Membrane Processes*, Springer

Pesch, H. J., Plail, M. (2012). "The Cold War and the Maximum Principle of Optimal Control", *Documenta Mathematica · Extra Volume ISMP*, 331–343.

Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) MATLAB. https://www.mathworks.com/help/matlab/ref/pchip.html, accessed at June 2017.

Polak, E. (1969). On primal and dual methods for solving discrete optimal control problems. In *2 Computing Methods in Optimization Problems*, Sep. 9-13, 1968; San Remo; Italy Academic Press Inc., New York

Price, K., Storn, R. (1997). Differential Evolution (DE). http://www1.icsi.berkeley.edu/~storn/code.html#cont, accessed at June 2017.

Price, K. V., Storn, R. M., Lampinen, J. A. (2005). *Differential evolution : a practical approach to global optimization*. Springer.

Rajesh, J., Gupta, K., Kusumakar, H. S., Jayaraman, V. K., Kulkarni, B. D. (2001). "Dynamic optimization of chemical processes using ant colony framework", *Computers & Chemistry*, *25*(6), 583–595.

Rao, A. V. (2009). "A survey of numerical methods for optimal control", *Advances in the Astronautical Sciences*, *135*(1), 497–528.

Ray, W. H. (1981). *Advanced Process Control*. McGraw-Hill.

Reddy, K. V., Husain, A. (1981). "Computation of optimal control policy with singular subarc", *The Canadian Journal of Chemical Engineering*, *59*(4), 557–559.

Renfro, J. G., Morshedi, A. M., Asbjornsen, O. A. (1987). "Simultaneous optimization and solution of systems described by differential/algebraic equations", *Computers & Chemical Engineering*, *11*(5), 503–517.

Rutquist, P. E., Edvall, M. M. (2010). PROPT - Matlab Optimal Control Software. *Tomlab Optimization Inc*, *260*.

Salomon, D. (2008). *A concise introduction to data compression*. Springer.

Sargent, R., Sullivan, G. R. (1978). "The development of an efficient optimal control package" In J. Stoer (Ed.), *Proceedings of the Eighth IFIP Conference on Optimization Techniques*. Heidelberg: Springer.

Sargent, R. W. H. (2000). "Optimal control", *Journal of Computational and Applied Mathematics*, *124*(1), 361–371.

Schlegel, M., Stockmann, K., Binder, T., Marquardt, W. (2005). "Dynamic optimization using adaptive control vector parameterization", *Computers & Chemical Engineering*, *29*(8), 1731–1751.

Shampine, L. F., Gladwell, I., Thompson, S. (2003). *Solving ODEs with MATLAB*. Cambridge University Press.

Spangelo, I. (1994). *Trajectory Optimization for Vehicles Using Control Vector Parameterization and Nonlinear Programming,* Ph.D. Thesis, Department of Engineering Cybernetics The Norwegian Institute of Technology.

Srinivasan, B., Palanki, S., Bonvin, D. (2003). "Dynamic optimization of batch processes I. Characterization of the nominal solution", *Computers and Chemical Engineering*, *27*(1), 1–26.

Storn, R., Price, K. (1995). "Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces". *International Computer Science Institute*.

Strang, G. (1999). "The Discrete Cosine Transform", *SIAM Review*, *41*(1), 135–147.

Subchan, S., Żbikowski, R. (2009). *Computational optimal control : tools and practice*. Wiley.

Sussmann, H. J., Willems, J. C. (1997). "300 years of optimal control: from the brachystochrone to the maximum principle", *IEEE Control Systems Magazine*, *17*(3), 32–44.

Tanartkit, P., Biegler, L. (1995). "Stable Decomposition for Dynamic Optimization", *Industrial & Engineering Chemistry Fundamentals*, *34*, 1253–1266.

Tanartkit, P., Biegler, L. T. (1997). "A nested, simultaneous approach for dynamic optimization problems—II: the outer problem", *Computers & Chemical Engineering*, *21*(12), 1365–1388.

Tsang, T. H., Himmelblau, D. M., Edgar, T. F. (1975). "Optimal control via collocation and non-linear programming.", *International Journal of Control*, *21*(5), 763–768.

Upreti, S. R. (2012). *Optimal Control for Chemical Engineers.* CRC Pr I Llc.

Vassiliadis, V. (1993). *Computational solution of dynamic optimization problems with general differential-algebraic constraints,* Ph.D. Thesis, Imperial College London (University of London)

Vassiliadis, V., Sargent, R. W. H., Pantelides, C. C. (1994). "Solution of a Class of Multistage Dynamic Optimization Problems. 1. Problems without Path Constraints", *Industrial & Engineering Chemistry Research*, *33*(9), 2111–2122.

von Stryk, O. (1999). Userguide for DIRCOL: A direct collocation method for the numerical solution of optimal control problems. *Lehrstuhl Für Höhere Mathematik Und Numerische*, (November).

von Stryk, O., Bulirsch, R. (1992). "Direct and indirect methods for trajectory optimization", *Annals of Operations Research*, *37*(1), 357–373.

APPENDIX A: MATLAB CODES USED

Appendix A.1. MATLAB Code Used in the CVO Method

```
%Van Der Pol Oscillator Problem
function MAIN
echo off; clc; clear all; close all; warning off; format shortG;
tic
global uLB uUB tf X0
global Ntgrid Ndctc tgrid InterpType
global optionsODE
global iplotu
global umin umax iplot
global icallode
icallode=0;
uLB=-0.3; uUB=1; tf=5; X0 = [0 1 0];


iplotu=0;


optionsODE = odeset('RelTol',1e-10,'AbsTol',1e-8);


InterpType = 'pchip'; % 'nearest'/'next'/'previous': for piecewise constant u's
                      % 'linear'/'pchip'/'spline'  : for continuous u's


%--- Solution Grid
Ntgrid = 11
tgrid = linspace(0,tf,Ntgrid);
%-END- Solution Grid


%--- Initial Guesses for the Optimizer
Umean=(uLB+uUB)/2;
pars0=[Umean*ones(1,Ntgrid)];
% pars0=[zeros(1,Ntgrid)];
Number_Of_Decision_Variables=length(pars0)
%-END- Initial Guesses for the Optimizer


LB = [-0.3*ones(length(pars0),1)]; UB =[ 1*ones(length(pars0),1)];
%
options = optimset('MaxFunEvals',50000,'MaxIter',5000,'Algorithm','sqp','HessUpdate','bfgs',...
                   'Display','iter','TolFun',1.E-10,'TolX',1.E-8,'FinDiffType','central','FinDiffRelStep',1.E-2,...
                   'PlotFcns',{@optimplotx,@optimplotfunccount,@optimplotfval})
[pars OBJFCT,exitflag,output] = "fmincon"(@ODEfct, pars0, [],[],[],[],LB,UB,[],options)
pars0=pars;
NP=20; IterMax=2500; TolF=1.E-8; CR=0.9; F=0.8; Strategy=6; Refresh=10; Ifplot=0;
[pars,OBJFCT] = devec3(@ODEfct,-inf,pars0,LB',UB',NP,IterMax,TolF,F,CR,Strategy,Refresh,Ifplot)
% pars0=pars;
icallode



%--- Call the Model After Optimization (for presentation purposes)
ugrid = Parameterization(pars, Ntgrid, Ndctc);
[t x]=ode15s(@ODEs,[tgrid],X0,optionsODE, pars)
FigHandle = figure;
unit = FigHandle.Units;
FigHandle.Units = 'centimeters';
set(FigHandle, 'Position', [2, 5, 15.5, 6.82]);
subtightplot(1,2,1,[0.08,0.1],[0.17,0.02],[0.075,0])
box on
line_fewer_markers(t,x(:,1), 11, 'ko-','spacing','curve','LockOnMax',1,'markersize',5);
line_fewer_markers(t,x(:,2), 11, 'k^-','spacing','curve','LockOnMax',1,'markersize',5);
line_fewer_markers(t,x(:,3), 11, 'kp-','spacing','curve','LockOnMax',1,'markersize',5);


legend('x_1','x_2','x_3','Location','Best');
set(gca,'FontSize',10)
set(gca,'FontName','Times New Roman')
ylabel('states (x(t))','FontName','Times New Roman','FontSize',10);
```

```
xlabel('t (h)','FontName','Times New Roman','FontSize',10);
xlim([0 tf]),ylim([-1 3]);grid on
subtightplot(1,2,2,[0.08,0.1],[0.17,0.02],[0.06,0.02])


plot(t,u,'k'); grid on
hold on
plot(tgrid,ugrid,'k.'); grid on
box on
set(gca,'FontSize',10)
set(gca,'FontName','Times New Roman')
ylabel('control (u(t))','FontName','Times New Roman','FontSize',10);
xlabel('t (h)','FontName','Times New Roman','FontSize',10);
xlim([0 tf]),ylim([-0.5 1.2]);grid on


umax=max(ugrid)
umin=min(ugrid)
PenaltyUpLow = 10.*( (max(0, uLB-umin) + (max(0, umax-uUB))))


PenaltyTotal = PenaltyUpLow
Objective = x(end,3)


ObjODE = Objective+PenaltyTotal


x(end,:)
toc
diary off
end % End of Main Program


% ======
function ObjODE=ODEfct(pars)
% Optimzer's Obj Fct Routine which integrates the ODEs to evaluate the ObjFct Value
global X0 tf uLB uUB
global tgrid Ntgrid Ndctc
global optionsODE
global ugrid
global iplotu


ugrid = Parameterization(pars, Ntgrid, Ndctc);


%--- Intermediate Plot
if (rem(iplotu,100) == 0)
sfigure(35);
clf(35)
plot(tgrid,ugrid,'r.-'); grid on;
legend('u','Location','Best');
drawnow;
end
iplotu=iplotu+1;
%-END- Intermediate Plot



umax=max(ugrid); umin=min(ugrid);


[t x]=ode15s(@ODEs,tgrid,X0,optionsODE, pars);


PenaltyUpLow = 10.*( (max(0, uLB-umin) + (max(0, umax-uUB))));


PenaltyTotal = PenaltyUpLow;


Objective = x(end,3);
ObjODE = Objective + PenaltyTotal;


end



% ======
```

```
function dydt = ODEs(t,x, pars)
global tgrid InterpType
global ugrid
global icallode
G=griddedInterpolant(tgrid,ugrid,InterpType); u=G(t);

dydt = [ (1-x(2).^2)*x(1)-x(2)+u
         x(1)
         x(1).^2+x(2).^2+u.^2];
icallode=icallode+1;

end

% ======
function U = Parameterization(pars, Ntgrid, Ndctc)
global tgrid iplotu
U=pars;
end
```

Appendix A.2. MATLAB Code Used in the CVP Method

```
%--- VanDerPol Oscillator Problem

function MAIN
echo off; clc; clear all; close all; warning off; format shortG;
tic

global uLB uUB tf X0
global optionsODE
global iplotu

uLB=-0.3; uUB=1; tf=5; X0 = [0 1 0];
iplotu=0;
optionsODE = odeset('RelTol',1e-10,'AbsTol',1e-8);

%--- Initial Guesses for the Optimizer
pars0=zeros(1,6); % initial guess
Number_Of_Decision_Variables=length(pars0)
%-END- Initial Guesses for the Optimizer

LB = [-20*ones(length(pars0),1)]; UB = [+20*ones(length(pars0),1)];
options = optimset('MaxFunEvals',5000,'MaxIter',500,'Algorithm','sqp',...
                'Display','iter','TolFun',1.E-10,'TolX',1.E-8,'FinDiffType','central')

pars OBJFCT,exitflag,output] = "fmincon"(@ODEfct, pars0, [],[],[],[],LB,UB,[],options)
pars0=pars;
NP=20; IterMax=2000; TolF=1.E-8; CR=0.9; F=0.8; Strategy=6; Refresh=10; Ifplot=0;
[pars,OBJFCT] = devec3(@ODEfct,-inf,pars0,LB',UB',NP,IterMax,TolF,F,CR,Strategy,Refresh,Ifplot)
% pars0=pars;

%--- Call the Model After Optimization (for presentation purposes)
[t x]=ode15s(@ODEs,[0 tf],X0,optionsODE, pars);
u  = Parameterization(pars, t);

FigHandle = figure;
unit = FigHandle.Units;
FigHandle.Units = 'centimeters';
set(FigHandle, 'Position', [2, 5, 15.5, 6.82]);
subtightplot(1,2,1,[0.08,0.1],[0.17,0.02],[0.075,0])
box on
line_fewer_markers(t,x(:,1), 11, 'ko-','spacing','curve','LockOnMax',1,'markersize',5);
line_fewer_markers(t,x(:,2), 11, 'k^-','spacing','curve','LockOnMax',1,'markersize',5);
line_fewer_markers(t,x(:,3), 11, 'kp-','spacing','curve','LockOnMax',1,'markersize',5);

legend('x_1','x_2','x_3','Location','Best');
set(gca,'FontSize',10)
```

```
set(gca,'FontName','Times New Roman')
ylabel('states (x(t))','FontName','Times New Roman','FontSize',10);
xlabel('t (h)','FontName','Times New Roman','FontSize',10);
xlim([0 tf]),ylim([-1 3]);grid on
subtightplot(1,2,2,[0.08,0.1],[0.17,0.02],[0.06,0.02])
plot(t,u,'k'); grid on
box on
set(gca,'FontSize',10)
set(gca,'FontName','Times New Roman')
ylabel('control (u(t))','FontName','Times New Roman','FontSize',10);
xlabel('t (h)','FontName','Times New Roman','FontSize',10);
xlim([0 tf]),ylim([-0.5 1.2]);grid on


umax=max(u)
umin=min(u)
PenaltyUpLow = 10.*( (max(0, uLB-umin) + (max(0, umax-uUB))))
PenaltyTotal = PenaltyUpLow
Objective = x(end,3)
ObjODE = Objective+PenaltyTotal
x(end,:)
toc
end % End of Main Program


% ======
function ObjODE=ODEfct(pars)
% Optimzer's Obj Fct Routine which integrates the ODEs to evaluate the ObjFct Value
global X0 tf uLB uUB
global optionsODE
global iplotu


[t x]=ode15s(@ODEs,[0 tf],X0,optionsODE, pars);
u  = Parameterization(pars, t);


%--- Intermediate Plot
if (rem(iplotu,100) == 0)
sfigure(45);
clf(45)
plot(t,u,'r.-'); grid on;
legend('u','Location','Best');
drawnow;
end
iplotu=iplotu+1;
%-END- Intermediate Plot


umax=max(u) ;
umin=min(u);
PenaltyUpLow = 10.*( (max(0, uLB-umin) + (max(0, umax-uUB))));
PenaltyTotal = PenaltyUpLow;
Objective = x(end,3);
ObjODE = Objective+PenaltyTotal;
end


% ======
function dydt = ODEs(t,x, pars)
 u = Parameterization(pars, t);
 dydt = [ (1-x(2).^2)*x(1)-x(2)+u
         x(1)
         x(1).^2+x(2).^2+u.^2];
end
% ======


function U = Parameterization(pars, t)
global  iplotu
% U = pars(1)*pars(2)*exp(pars(3).*t);          % i) Constant + exponential function
% U = pars(1)*exp(pars(2).*t); ;                % ii) constant + exponential term
% U = pars(1) + +pars(2)*t+pars(3)*exp(-pars(4).*t);   % iii) Polynomial + Constant + Exponential function
% U = pars(1) + pars(2)*t + pars(3)*t.^2;             % iv) 2nd order polynomial
% U = pars(1) + pars(2)*t + pars(3)*t.^2 +pars(4)*t.^3 % v) 3rd order polynomial
% U = pars(1) + pars(2)*t + pars(3)*t.^2 +pars(4)*t.^3+pars(5)*t.^4 % vi) 4th order polynomial
U = pars(1) + pars(2)*t + pars(3)*t.^2 +pars(4)*t.^3+pars(5)*t.^4+pars(6)*t.^5% vii) 5th order polynomial
```

```
end
```

Appendix A.3. MATLAB Code Used in the CVP-DCT Method

```
%--- VanDerPol Oscillator Problem

function MAIN
echo off; clc; clear all; close all; warning off; format shortG;
tic
global uLB uUB tf X0
global Ntgrid Ndctc tgrid InterpType
global optionsODE
global iplotu
global umin umax iplot
global icallode
icallode=0;
uLB=-0.3; uUB=1; tf=5; X0 = [0 1 0];

iplotu=0;

optionsODE = odeset('RelTol',1e-10,'AbsTol',1e-8);



InterpType = 'pchip'; % 'nearest'/'next'/'previous': for piecewise constant u's
                      % 'linear'/'pchip'/'spline'  : for continuous u's



%--- Solution Grid
%    For CVI-Direct set Ndctc=Ntgrid & Nzones=1
%    Ntgrid must be divisible with Nzones
Ntgrid = 11
Ndctc  = 5
tgrid = linspace(0,tf,Ntgrid);
%-END- Solution Grid

%--- Initial Guesses for the Optimizer
Umean=(uLB+uUB)/2;
pars0=[dct(Umean*(1,Ntgrid))];
% pars0=[dct(zeros*(1,Ntgrid))];
pars0=[pars0(1:Ndctc)]
Number_Of_Decision_Variables=length(pars0)
%-END- Initial Guesses for the Optimizer



% pars0=pars;
LB = [-20*ones(length(pars0),1)]; UB = [+20*ones(length(pars0),1)];

options = optimset('MaxFunEvals',50000,'MaxIter',5000,'Algorithm','sqp','HessUpdate','bfgs',...
                   'Display','iter','TolFun',1.E-10,'TolX',1.E-8,'FinDiffType','central','FinDiffRelStep',1.E-2,...
                   'PlotFcns',{@optimplotx,@optimplotfunccount,@optimplotfval})
[pars OBJFCT,exitflag,output] = "fmincon"(@ODEfct, pars0, [],[],[],[],LB,UB,[],options)
pars0=pars;
NP=20; IterMax=2500; TolF=1.E-8; CR=0.9; F=0.8; Strategy=6; Refresh=10; Ifplot=0;
[pars,OBJFCT] = devec3(@ODEfct,-inf,pars0,LB',UB',NP,IterMax,TolF,F,CR,Strategy,Refresh,Ifplot)
% pars0=pars;
icallode
%--- Call the Model After Optimization (for presentation purposes)
ugrid = Parameterization(pars, Ntgrid, Ndctc);
[t x]=ode15s(@ODEs,[0 tf],X0,optionsODE, pars)
G=griddedInterpolant(tgrid,ugrid,InterpType); u=G(t);

FigHandle = figure;
unit = FigHandle.Units;
FigHandle.Units = 'centimeters';
```

```
set(FigHandle, 'Position', [2, 5, 15.5, 6.82]);
subtightplot(1,2,1,[0.08,0.1],[0.17,0.02],[0.075,0])
box on
line_fewer_markers(t,x(:,1), 11, 'ko-','spacing','curve','LockOnMax',1,'markersize',5);
line_fewer_markers(t,x(:,2), 11, 'k^-','spacing','curve','LockOnMax',1,'markersize',5);
line_fewer_markers(t,x(:,3), 11, 'kp-','spacing','curve','LockOnMax',1,'markersize',5);

legend('x_1','x_2','x_3','Location','Best');
set(gca,'FontSize',10)
set(gca,'FontName','Times New Roman')
ylabel('states (x(t))','FontName','Times New Roman','FontSize',10);
xlabel('t (h)','FontName','Times New Roman','FontSize',10);
xlim([0 tf]),ylim([-1 3]);grid on
subtightplot(1,2,2,[0.08,0.1],[0.17,0.02],[0.06,0.02])



plot(t,u,'k'); grid on
hold on
plot(tgrid,ugrid,'k.'); grid on
box on
set(gca,'FontSize',10)
set(gca,'FontName','Times New Roman')
ylabel('control (u(t))','FontName','Times New Roman','FontSize',10);
xlabel('t (h)','FontName','Times New Roman','FontSize',10);
xlim([0 tf]),ylim([-0.5 1.2]);grid on

umax=max(ugrid)
umin=min(ugrid)
PenaltyUpLow = 10.*( (max(0, uLB-umin) + (max(0, umax-uUB))))
PenaltySmooth = 0.001*(norm(diff(u,2)));



PenaltyTotal = PenaltyUpLow + 0*PenaltySmooth
Objective = x(end,3)

ObjODE = Objective+PenaltyTotal

x(end,:)
toc
end % End of Main Program

% ======
function ObjODE=ODEfct(pars)
% Optimzer's Obj Fct Routine which integrates the ODEs to evaluate the ObjFct Value
global X0 tf uLB uUB
global tgrid Ntgrid Ndctc
global optionsODE
global ugrid
global iplotu

ugrid = Parameterization(pars, Ntgrid, Ndctc);

%--- Intermediate Plot
if (rem(iplotu,100) == 0)
sfigure(35);
clf(35)
plot(tgrid,ugrid,'r.-'); grid on;
legend('u','Location','Best');
drawnow;
end
iplotu=iplotu+1;
%-END- Intermediate Plot



umax=max(ugrid); umin=min(ugrid);

[t x]=ode15s(@ODEs,tgrid,X0,optionsODE, pars);
```

```
PenaltyUpLow = 10.*( (max(0, uLB-umin) + (max(0, umax-uUB))));

PenaltyTotal = PenaltyUpLow;

Objective = x(end,3);
ObjODE = Objective + PenaltyTotal;

end



% ======
function dydt = ODEs(t,x, pars)
global tgrid InterpType
global ugrid
global icallode
G=griddedInterpolant(tgrid,ugrid,InterpType); u=G(t);

dydt = [ (1-x(2).^2)*x(1)-x(2)+u
         x(1)
         x(1).^2+x(2).^2+u.^2];
icallode=icallode+1;
end

% ======
function U = Parameterization(pars, Ntgrid, Ndctc)
global tgrid iplotu
U=idct(pars(1:Ndctc),Ntgrid); % for parameterization without constant at each element
end
```