Optimization of Linear Time-invariant Dynamic
Systems Without Lagrange Multipliers

A Thesis Presented to
The Faculty of the Russ College of Engineering and Technology
Ohio University

In partial Fulfillment of the
Requirements for the Degree
Master of Science

By
Tawiwat Veeraklaew
Department of Mechanical Engineering
November 1995
Acknowledgment

The author believes that this thesis could be done only because of these persons whom the author respects and loves. Therefore, the author wants to thank them. First, Dr. Sunil K. Agrawal is the author’s advisor. He is a very kind and helpful professor. Second, all professors in the Mechanical Engineering Department at Ohio University and third, my teachers at the Chulachomklao Royal Military Academy for providing scholarship. Finally, the author’s family, Passarawon-Artima-Taweeiya Veeraklaew for their constant encouragement and support.
Abstract

This thesis describes a new technique for optimization of linear time-invariant dynamic systems with \( n \) states and \( m \) control inputs commanded to move between two fixed states in a prescribed final time where Lagrange multipliers are not needed in the solution procedure. In the literature, this problem is solved using Lagrange multipliers. These multipliers are artificial variables and their values are unknown at initial and final time. In this new procedure, a linear transformation is used to express the state and the control vector in terms of higher derivatives of a subset of state variables. On substituting these expressions in the cost functional, the problem becomes an unconstrained optimization problem and the functional contains higher derivatives of some of the states. A variational statement of this new cost functional is written and the necessary conditions for optimality are obtained. These conditions are then solved using classical weighted residual methods. It has been shown in this thesis that this new procedure, solving higher-order differential equations using weighted-residual technique, requires matrix computations of dimension \( m \) as opposed to conventional techniques which require matrix computations of dimension \( n \).
# Table of contents

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
</tr>
<tr>
<td>Abstract</td>
</tr>
<tr>
<td>Table of contents</td>
</tr>
<tr>
<td>Chapter 1 Introduction</td>
</tr>
<tr>
<td>1.1 Background</td>
</tr>
<tr>
<td>1.2 Optimization for Linear Dynamic Systems</td>
</tr>
<tr>
<td>1.2.1 Problem Statement</td>
</tr>
<tr>
<td>1.2.2 Necessary Conditions</td>
</tr>
<tr>
<td>1.3 Matrix Exponential Approach</td>
</tr>
<tr>
<td>1.4 Outline of the Thesis</td>
</tr>
<tr>
<td>Chapter 2 Optimization Problems for the Case ($n = 2m$)</td>
</tr>
<tr>
<td>2.1 Introduction</td>
</tr>
<tr>
<td>2.2 4th-order TPBVP Formulation</td>
</tr>
<tr>
<td>2.2.1 Elimination of Lagrange Multipliers</td>
</tr>
<tr>
<td>2.2.2 Alternative Variational Form</td>
</tr>
<tr>
<td>2.3 Weighted Residual Methods</td>
</tr>
<tr>
<td>2.4 Galerkin's Solution</td>
</tr>
<tr>
<td>2.4.1 Mode Function: A Particular choice</td>
</tr>
<tr>
<td>2.5 Examples</td>
</tr>
<tr>
<td>2.5.1 A 2 Degree-of-freedom system</td>
</tr>
</tbody>
</table>
2.5.2 A Four Degree-of-freedom system

Chapter 3 Optimization Problems for the Case ($n = pm$)

3.1 Introduction

3.2 State Transformation

3.3 Optimality Conditions

3.4 Weighted Residual Solution

3.4.1 Mode Functions

3.4.2 Recursion with mode

3.4.3 Overall Computation

3.5 Illustrative Example

3.6 Summary

Chapter 4 Optimization Problem for the Case ($n = rm$)

4.1 Introduction

4.2 State Transformation

4.2.1 State Variables as Higher Derivatives

4.2.2 Boundary Conditions

4.3 Optimization

4.3.1 Variational Statement

4.3.2 Differential Equation

4.3.3 Boundary Terms

4.4 Weighted Residual Solution
Chapter 1

Introduction

1.1 Background

The objective of this chapter is to explain the necessary theory for optimization of dynamic systems using Lagrange multipliers. This will give the necessary background for presenting the procedure for optimization of dynamic systems without Lagrange multipliers.

1.2 Optimization Problem for Linear Dynamic Systems

This section discusses the optimization problem for linear time-invariant dynamic systems with fixed end time and end point.

1.2.1 Problem Statement

The statement of the problem is to choose the path \( X(t) \) that will take the system from an initial state \( X_0 \) at time \( t_0 \) to a final state \( X_f \) at time \( t_f \). During the path from \( t_0 \) to \( t_f \), the following quadratic functional dependent on state variables and control input must be minimized

\[
J = \frac{1}{2} \int_{t_0}^{t_f} [X^T Q X + U^T R U] dt
\]  

(1.1)
where $Q$ is a symmetric matrix and $R$ is a symmetric and positive definite matrix.

1.2.2 Necessary Conditions

The dynamic equations are typically written in the following form:

$$\dot{X} = AX + BU$$

(1.2)

where $A$ is an $(n \times n)$ matrix, $X$ is an $n$-dimensional vector, $B$ is an $(n \times m)$ matrix, and $U$ is a $(m \times 1)$ vector of control inputs. According to conventional dynamic optimization theory, the optimal solution must satisfy the state equations (1.2), the following $n$ costate equations (1.3), and $m$ optimality equations (1.4)

$$\dot{\Lambda} = -QX - AT\Lambda$$

(1.3)

$$RU + B^T\Lambda = 0$$

(1.4)

where $\Lambda$ is an $(n \times 1)$ vector of Lagrange multipliers. On solving $U = -R^{-1}B^T\Lambda$ from Eq.(1.4) and substituting in Eq.(1.2), the necessary conditions become

$$\dot{X} = AX - BR^{-1}B^T\Lambda$$

(1.5)

$$\dot{\Lambda} = -QX - AT\Lambda$$

(1.6)

with boundary conditions specified on $X$ at $t_0$ and $t_f$. This problem is called a Two-Point Boundary Value Problem (TPBVP). The above $2n$ first-order differential equations can be solved by conventional matrix exponential technique or shooting techniques.
1.3 Matrix Exponential Approach

In this section, the solution is described where $A$, $B$, $R$, and $Q$ are time-invariant matrices. The Eqs. (1.5) and (1.6) can be put together in the following form:

\[
\begin{bmatrix}
\dot{X} \\
\dot{\Lambda}
\end{bmatrix} =
\begin{bmatrix}
A & -BR^{-1}B^T \\
-Q & -A^T
\end{bmatrix}
\begin{bmatrix}
X \\
\Lambda
\end{bmatrix}
\]  

(1.7)

On defining \[
\begin{bmatrix}
A & -BR^{-1}B^T \\
-Q & -A^T
\end{bmatrix} = \hat{A}
\], the solution of Eq.(1.7) is

\[
\begin{bmatrix}
X \\
\Lambda
\end{bmatrix}_{(t_0)} = e^{\hat{A}(t-t_0)}
\begin{bmatrix}
X \\
\Lambda
\end{bmatrix}_{(t_0)}
\]  

(1.8)

Rewriting the matrix exponential of Eq.(1.8) in the following way

\[
e^{\hat{A}(t-t_0)} =
\begin{bmatrix}
S_{11}(t-t_0) & S_{12}(t-t_0) \\
S_{21}(t-t_0) & S_{22}(t-t_0)
\end{bmatrix}
\]  

(1.9)

and on substituting Eq.(1.9) into Eq.(1.8)

\[
\begin{bmatrix}
X \\
\Lambda
\end{bmatrix}_{(t)} =
\begin{bmatrix}
S_{11}(t-t_0) & S_{12}(t-t_0) \\
S_{21}(t-t_0) & S_{22}(t-t_0)
\end{bmatrix}
\begin{bmatrix}
X \\
\Lambda
\end{bmatrix}_{(t_0)}
\]  

(1.10)

On evaluating the above equation at $t = t_f$, Eq.(1.10) becomes

\[
\begin{bmatrix}
X_f \\
\Lambda_f
\end{bmatrix} =
\begin{bmatrix}
S_{11}(t_f-t_0) & S_{12}(t_f-t_0) \\
S_{21}(t_f-t_0) & S_{22}(t_f-t_0)
\end{bmatrix}
\begin{bmatrix}
X_0 \\
\Lambda_0
\end{bmatrix}
\]  

(1.11)

Given $X_f$ and $X_0$, $\Lambda_0$ can be computed from the equation. The expression is

\[
\Lambda_0 = \left[S_{12}(t_f-t_0)\right]^{-1}
\begin{bmatrix}
X_f - S_{11}(t_f-t_0)X_0
\end{bmatrix}
\]  

(1.12)
Then, if \( S_{12}(t_f - t_i) \) is nonsingular, the solution of \( X(t) \) can be plotted from Eq.(1.8). If \( A, B, R, \) and \( Q \) are time varying matrices, other suitable techniques such as shooting techniques are used to find the solution.

### 1.4 Outline of the Thesis

The next three chapters discuss a new solution technique for dynamic optimization of linear time-invariant dynamic systems. This thesis describes a new procedure where Lagrange multipliers are not needed in the analysis. The following three kinds of problems are addressed in this thesis.

(i) \( n = 2m \) \hspace{1cm} (Chapter 2)  

(ii) \( n = pm \) where \( p \) is an integer. \hspace{1cm} (Chapter 3)  

(iii) \( n = rm \) where \( r \) is an arbitrary choice. \hspace{1cm} (Chapter 4)
Chapter 2
Optimization Problems for the Case \(( n = 2m )\)

2.1 Introduction

This chapter deals with dynamic optimization of vibrating spring-mass-damper systems which have one actuator input for every degree-of-freedom and are characterized by \( m \) second-order linear coupled differential equations with constant coefficient. For this system the number of states \( n = 2m \). In this chapter, a different form of the variational statement is presented where instead of \( 2n \) first-order differential equations that characterize the optimal trajectory, the system is described by \( m \) fourth-order differential equations in the state variables. In this new form, the costate variables do not appear explicitly in the optimality equations. As a result, the problem becomes highly attractive to solve using classical weighted residual methods. The merit of this method is that the convergence is achieved with a very small number of modes.

The dynamic model of an \( m \) degree-of-freedom mechanical system consisting of springs, mass, and dampers can be written as

\[
M \ddot{q} + C \dot{q} + Kq = U
\]

(2.1)

where \( M, C, \) and \( K \) are \(( m \times m )\) symmetric matrices usually referred to as the mass, damping, and stiffness matrices. Also, \( M \) is a positive definite matrix. \( q \) is an \( m \)-dimensional vector of generalized coordinates. \( U \) is an \( m \)-dimensional vector of actuator inputs and ‘dots’ denote the derivatives of variables with respect to time.
2.2 4th-order TPBVP Formulation

The equation (2.1) are \( m \) second order differential equations in the generalized coordinates and can be reduced to \( n = 2m \) first order differential equations in the following way:

\[
\begin{align*}
\dot{q} &= \ddot{q} \\
\ddot{q} &= M^{-1}(U - C\ddot{q} - Kq)
\end{align*}
\]  

(2.2)

The statement of the problem is to choose the path \( q(t) \) that will take the system from an initial state \( q_0 \) at time \( t_0 \) to a final state \( q_f \) at time \( t_f \). During the path from \( t_0 \) to \( t_f \), the following quadratic functional dependent on state variables and control input must be minimized

\[
J = \frac{1}{2} \int_{t_0}^{t_f} \left[ q^T Q_1 q + \dot{q}^T Q_2 \dot{q} + U^T RU \right] dt
\]  

(2.3)

where \( Q_1, Q_2 \) are symmetric matrices and \( R \) is a symmetric and positive definite matrix.

According to conventional dynamic optimization theory, the optimal solution must satisfy the \( n \) state equations (2.2), the following \( n \) costate equations (2.4), and \( m \) optimality equations (2.5)

\[
\begin{align*}
\dot{\lambda} &= -Q_1 \lambda + KM^{-1} \dot{\lambda} \\
\dot{\lambda} &= -Q_2 \ddot{q} - \lambda + CM^{-1} \dot{\lambda} \\
RU + M^{-1} \dot{\lambda} &= 0
\end{align*}
\]  

(2.4) (2.5)
where $\lambda$ and $\overline{\lambda}$ are Lagrange multipliers. On solving for $U = -R^{-1}M^{-1}\overline{\lambda}$ from Eq.(2.5) and substituting in Eq.(2.2), the necessary conditions can be written in the matrix form:

$$
\begin{bmatrix}
\dot{q} \\
\dot{\overline{q}} \\
\dot{\lambda} \\
\dot{\overline{\lambda}}
\end{bmatrix} =
\begin{bmatrix}
0 & I & 0 & 0 \\
-M^{-1}K & -M^{-1}C & 0 & -M^{-1}R^{-1}M^{-1} \\
-Q_1 & 0 & 0 & KM^{-1} \\
0 & -Q_2 & -I & CM^{-1}
\end{bmatrix}
\begin{bmatrix}
q \\
\overline{q} \\
\lambda \\
\overline{\lambda}
\end{bmatrix} 
$$ (2.6)

In summary, the dynamic model of an $m$ degree-of-freedom mechanical system is Eq.(2.1). The system is characterized by $n$ first-order differential equations Eq.(2.2). For this system, $n = 2m$, i.e., the system has one actuator input for every degree-of-freedom. It will be shown in this section that an alternative form of the optimality equations can be written in which the state variables appear in their fourth derivative and the costate variables are eliminated. This can be achieved by adopting either of the two procedures: (i) eliminating the costate variables from Eq.(2.6), (ii) reworking the variational form without Lagrange multipliers by substituting $U$ from Eq.(2.1) in Eq.(2.3). It will be shown in section 2.2.1 that both procedures lead to the same result.

### 2.2.1 Elimination of Lagrange Multipliers

From Eq.(2.6), the individual equations can be written below

$$
\dot{q} = \overline{q} 
$$ (2.7)

$$
\dot{\overline{q}} = -M^{-1}Kq - M^{-1}C\overline{q} - M^{-1}R^{-1}M^{-1}\overline{\lambda} 
$$ (2.8)

$$
\dot{\lambda} = -Q_1q + KM^{-1}\overline{\lambda} 
$$ (2.9)
\[ \ddot{\lambda} = -Q_2 \ddot{q} - \dot{\lambda} + CM^{-1} \lambda \] (2.10)

On substituting \( q \) from Eq.(2.7) in Eq.(2.8), it becomes a second-order differential equation

\[ \ddot{q} = -M^{-1}C \dot{q} - M^{-1}K \dot{q} - M^{-1}R^{-1}M^{-1} \lambda \] (2.11)

On taking the time derivative of Eq.(2.10)

\[ \dddot{\lambda} = -Q_2 \dddot{q} - \dddot{\lambda} + CM^{-1} \ddot{\lambda} \] (2.12)

On rearranging Eq.(2.12) and substituting \( \dot{\lambda} \) from Eq.(2.9)

\[ \dddot{\lambda} - CM^{-1} \ddot{\lambda} - Q_1 \dot{q} + KM^{-1} \lambda + Q_2 \dddot{q} = 0 \] (2.13)

From Eq.(2.11), we can find \( \lambda \) and take time derivatives to get \( \dot{\lambda} \), and \( \ddot{\lambda} \) in terms of variable \( q \) alone.

\[ \lambda = -\left[ M^{-1}R^{-1}M^{-1} \right]^{-1} \left[ \dddot{q} + M^{-1}C \dddot{q} + M^{-1}K \dddot{q} \right] \] (2.14)

\[ \dot{\lambda} = -\left[ M^{-1}R^{-1}M^{-1} \right]^{-1} \left[ \dddot{q} + M^{-1}C \dddot{q} + M^{-1}K \dddot{q} \right] \] (2.15)

\[ \ddot{\lambda} = -\left[ M^{-1}R^{-1}M^{-1} \right]^{-1} \left[ \dddot{q} + M^{-1}C \dddot{q} + M^{-1}K \dddot{q} \right] \] (2.16)

On substituting Eqs.(2.14), (2.15), (2.16) in Eq.(2.13), the optimality can be rewritten as

\[ S_4 \dddot{q} + S_3 \dddot{q} + S_2 \dddot{q} + S_1 \dot{q} + S_0 \dot{q} = 0 \] (2.17)

where \( S_i, \ i = 0, \ldots, 4 \) are \( (m \times m) \) matrices defined as
\[ S_4 = MRM \]
\[ S_3 = MRC - CRM \]
\[ S_2 = MKR + KRM - CRC - Q_2 \]
\[ S_1 = KRC - CRK \]
\[ S_0 = KRK + Q_1 \]  

\[ (2.18) \]

2.2.2 Alternative Variational Form

On substituting \( U \) from Eq.(2.1) into Eq.(2.3), the cost functional takes the following form:

\[ J = \int_{t_0}^{t_f} \frac{1}{2} [q^T \dot{q} + q^T \dot{Q}_2 \ddot{q} + (M \ddot{q} + C \dot{q} + Kq)^T R(M \ddot{q} + C \dot{q} + Kq)] dt \]

where end conditions on \( q_i \) and \( \dot{q}_i \) are prescribed at \( t_0 \) and \( t_f \). On further simplification, it reduces to

\[ J = \int_{t_0}^{t_f} \frac{1}{2} [\dot{q}^T MRM \ddot{q} + 2 \dot{q}^T MRC \dot{q} + 2 \dot{q}^T MRK q + 2 \dot{q}^T CRK q + \dot{q}^T (CRC + Q_2) \ddot{q} + \dot{q}^T (KRK + Q_1) q] dt \]

Mathematically, the form of this cost functional is

\[ J = \int_{t_0}^{t_f} F(q_1, \ldots, q_m, \dot{q}_1, \ldots, \dot{q}_m, \ddot{q}_1, \ldots, \ddot{q}_m) dt \]  

\[ (2.20) \]

The variation of this functional is

\[ \delta J = \int_{t_0}^{t_f} \sum_{i=1}^{m} \left( \frac{\partial F}{\partial q_i} \frac{d}{dt} \frac{\partial F}{\partial \dot{q}_i} + \frac{d^2}{dt^2} \frac{\partial F}{\partial q_i} \right) h_i \left. dt \right|_{t_0}^{t_f} + \sum_{i=1}^{m} \left( \frac{\partial F}{\partial \dot{q}_i} \frac{d}{dt} \frac{\partial F}{\partial q_i} \right) h_i \left. \right|_{t_0}^{t_f} + \sum_{i=1}^{m} \left( \frac{\partial F}{\partial \ddot{q}_i} \right) h_i \left. \right|_{t_0}^{t_f} \]  

\[ (2.21) \]
where \( h_i(t) \) are the variations of \( q_i(t) \). Since, \( q_i \) and \( \dot{q}_i \) are specified at end points, \( h_i \) and \( \dot{h}_i \) must vanish. Hence the necessary conditions for optimization are

\[
\frac{\partial F}{\partial q_i} \frac{d}{dt} \frac{\partial F}{\partial \dot{q}_i} + \frac{d^2}{dt^2} \frac{\partial F}{\partial \ddot{q}_i} = 0, \quad i = 1, \ldots, m
\]  

(2.22)

On using the expression of \( F \) from Eq.(2.19) and substituting in Eq.(2.22), it can be shown that

\[
\frac{\partial F}{\partial q} = KRM \ddot{q} + KRC \dot{q} + (KRK + Q_1)q
\]

\[
\frac{d}{dt} \frac{\partial F}{\partial \dot{q}} = CRM \dddot{q} + CRK \ddot{q} + (CRC + Q_2)\dddot{q}
\]

\[
\frac{d^2}{dt^2} \frac{\partial F}{\partial \ddot{q}} = MRM \dddot{q} + MRC \dddot{q} + MRK \dddot{q}
\]

where \([MRK]^T = KRM\), \([CRK]^T = KRC\), and \([MRC]^T = CRM\), and the equation (2.22) becomes

\[
S_4 \dddot{q} + S_3 \dddot{q} + S_2 \dddot{q} + S_1 \ddot{q} + S_0 q = 0
\]  

(2.23)

where \( S_i, \ i = 0, \ldots, 4 \) are \((m \times m)\) matrices defined as

\[
S_4 = MRM \\
S_3 = MRC - CRM \\
S_2 = MRK + KRM - CRC - Q_2 \\
S_1 = KRC - CRK \\
S_0 = KRK + Q_1
\]  

(2.24)

These are the same optimality conditions listed in Eqs.(2.17) and (2.18).
2.3 Weighted Residual Methods

Different forms of weighted residual methods have been used to solve boundary value problems. A summary of these methods is available in [1]. These methods can be classified as: (a) those that satisfy the differential equations approximately over the domain but satisfy the boundary conditions exactly such as Galerkin's method, method of moments, collocation method, and method of sub-regions, (b) weak formulations which satisfy the differential equations approximately over the domain and the boundary conditions only partially, and (c) boundary element methods which satisfy the differential equations exactly over the domain but boundary conditions only approximately such as Trefftz method.

Due to the nature of our optimization problem with fixed end points at $t_0$ and $t_f$, only the methods classified in category (a) were considered suitable. The underlying fundamental behind this method can be summarized using the following simple example. Consider the problem

$$\eta(q) - p = 0 \quad (2.25)$$

where $\eta(\cdot)$ is a differential operator, $q$ is a function of time, and $p$ is a constant. The solution $q(t)$ must satisfy the given boundary conditions at the initial and final time. In this method, $q(t)$ is approximated as

$$q(t) = \sum_{i=1}^{r} \alpha_i \phi_i(t) \quad (2.26)$$

where $\alpha_i$ are undetermined parameters and $\phi_i(t)$ are linearly independent mode functions selected from a complete set of functions such as a polynomial function. These functions
are usually chosen to satisfy admissibility conditions relating to the boundary conditions. On substituting Eq.(2.24) in Eq.(2.25), the following error function results

\[ \varepsilon(t) = \eta(q) - p \]  \hspace{1cm} (2.27)

This error function \( \varepsilon(t) \) is forced to be zero, in the average sense, by setting weighted integrals of the residual error equal to zero, i.e.,

\[ \int_{t_0}^{t_f} \varepsilon \psi_i dt = 0, \quad i = 1, \ldots, k \]  \hspace{1cm} (2.28)

where \( \psi_i(t) \) are the weighting functions. The category (a) methods differ primarily in their selection of weighting functions. For example, the method of moments uses weighting functions as \( t^i, \ i = 1, \ldots, k \), and Galerkin’s method uses weighting functions the same as mode functions. In this chapter, Galerkin’s method is selected to obtain the approximate solution of the problem because of its generality and ubiquitous use in solving problems of mechanics. The mode functions in this problem are chosen as polynomial due to their simplicity of analytical integration.

### 2.4 Galerkin’s Solution

The approximate solution of the fourth-order differential Eq.(2.17) must be obtained subjected to the following boundary conditions \( q(t_o) = q_o, \ q'(t_o) = q_o', \ q(t_f) = q_f, \) and \( q'(t_f) = q_f' \). In order to ensure admissibility, the approximate solution must have the following form [3]:

\[ q(t) = \Phi_o(t) + \sum_{i=1}^{k} L_i \phi_i(t) \]  \hspace{1cm} (2.29)
Where $\Phi_o(t)$ is an $m$-dimensional vector of mode functions that satisfies the boundary conditions of the vector $q$ at time $t_0$ and $t_f$. $\phi_i(t)$ are mode functions which vanish up to the first derivative at the two end points. As a result, $q(t)$ always satisfies the boundary conditions of the problem. $L_1, \ldots, L_k$ are $m$-dimensional constant vectors which are determined by minimizing the residual error. On substituting Eq.(2.29) in Eq.(2.17), the following error vector results:

$$
\varepsilon(t) = S_4 \dddot{\Phi}_0 + S_3 \dddot{\Phi}_0 + S_2 \ddot{\Phi}_0 + S_1 \dot{\Phi}_0 + S_0 \Phi_0
$$

$$
+ L_1 (S_4 \dddot{\phi}_1 + S_3 \dddot{\phi}_1 + S_2 \ddot{\phi}_1 + S_1 \dot{\phi}_1 + S_0 \phi_1)
$$

$$
+ L_2 (S_4 \dddot{\phi}_2 + S_3 \dddot{\phi}_2 + S_2 \ddot{\phi}_2 + S_1 \dot{\phi}_2 + S_0 \phi_2)
$$

$$
+ \ldots \ldots
$$

$$
+ L_k (S_4 \dddot{\phi}_k + S_3 \dddot{\phi}_k + S_2 \ddot{\phi}_k + S_1 \dot{\phi}_k + S_0 \phi_k)
$$

In accordance with Galerkin's procedure, the error function must be chosen to be orthogonal to the mode functions

$$
\int_0^{t_f} \varepsilon(t) \phi_i(t) \, dt = 0, \quad i = 1, \ldots, k
$$

(2.31)

This leads to $mk$ scalar equations which can be used to solve for the $mk$ elements of the vector $L_1, \ldots, L_k$. The equation (2.31) can be written in a matrix form:

$$
\begin{bmatrix}
T_{11} & T_{12} & \cdots & T_{1k} \\
T_{21} & T_{22} & \cdots & T_{2k} \\
\vdots & \vdots & \ddots & \vdots \\
T_{k1} & T_{k2} & \cdots & T_{kk}
\end{bmatrix}
\begin{bmatrix}
L_1 \\
L_2 \\
\vdots \\
L_k
\end{bmatrix}
= 
\begin{bmatrix}
-R_1 \\
-R_2 \\
\vdots \\
-R_k
\end{bmatrix}
$$

(2.32)

where $T_{pi}$ is a $(m \times m)$ matrix subblock, and $R_p$ is a $(m \times 1)$ vector defined below:
\[ T_{pl} = \int_{t_0}^{t_f} \left( S_4 \ddot{\phi}_1 + S_3 \ddot{\phi}_1 + S_2 \ddot{\phi}_1 + S_1 \dot{\phi}_1 + S_0 \dot{\phi}_1 \right) \phi_p dt \]

\[ R_p = \int_{t_0}^{t_f} \left( S_4 \ddot{\phi}_0 + S_3 \ddot{\phi}_0 + S_2 \ddot{\phi}_0 + S_1 \dot{\phi}_0 + S_0 \dot{\phi}_0 \right) \phi_p dt \]

(2.33)

The above equation can be inverted to solve for vectors \( L_1, \ldots, L_k \).

2.4.1 Mode Function: A Particular Choice

It is quite evident that any set of \( \Phi_0 \) and \( \phi_i(t) \) that satisfies admissibility conditions described in the last section is a valid set of mode functions. In this chapter, \( \Phi_0(t) \) is chosen as the following cubic function of time

\[ \Phi_0(t) = q_0 + \dot{q}_0 t + \frac{3}{t_f^2} (q_f - q_0) - \frac{2}{t_f} \dot{q}_0 - \frac{1}{t_f} \ddot{q}_f \] \( + [- \frac{2}{t_f^3} (q_f - q_0) + \frac{1}{t_f^2} (\dot{q}_f + \ddot{q}_0) ] t^3 \)

(2.34)

It can be easily verified that \( \Phi_0(t_0) = q_0, \Phi_0(t_f) = q_f, \dot{\Phi}_0(t_0) = \dot{q}_0, \) and \( \ddot{\Phi}_0(t_f) = \dddot{q}_f \).

The mode functions \( \phi_i(t) \) for \( t_0 = 0 \) are selected as

\[ \phi_i(t) = t^2 (t - t_f) t_i, \quad i = 1, \ldots, k \]

(2.35)

These mode functions possess the properties \( \phi_i(t_0) = \phi(t_f) = \dot{\phi}_i(t_0) = \dot{\phi}_i(t_f) = 0 \).

With these mode functions, the matrix \( T_{pl} \) and the right hand side vector \( R_p \) can be analytically computed. These are listed in Appendix A in Eq.(A1) and Eq.(A2) respectively.
2.5 Examples

2.5.1 A 2 degree-of-freedom system

A two degree-of-freedom spring-mass-damper system is used as an example. The system is sketched in Figure 1. The matrices $M$, $C$, and $K$ for this system are:

$$
M = \begin{bmatrix}
    m_1 & 0 \\
    0 & m_2
\end{bmatrix}
$$

$$
C = \begin{bmatrix}
    c_1 + c_2 & -c_2 \\
    -c_2 & c_2 + c_3
\end{bmatrix}
$$

$$
K = \begin{bmatrix}
    k_1 + k_2 & -k_2 \\
    -k_2 & k_2 + k_3
\end{bmatrix}
$$

The states are commanded to move from $q_0 = (10, 20)^T$, $\dot{q}_0 = (10, 20)^T$ to the equilibrium position with zero final velocity, i.e., $q_f = q_f = 0$. The parameters of Eq.(2.29) in MKS units are $m_1 = m_2 = 1.0$, $c_1 = c_3 = 1.0$, $c_2 = 2.0$, $k_1 = k_2 = k_3 = 3.0$. The matrices in the cost function are: $R = I_2$, $Q_1 = Q_2 = 0$.

Figure 2 compares the optimal response curves of the state variables $q_1$ and $q_2$ obtained using Matrix-exponential method and Galerkin's method for $t_0 = 0$ and $t_f = 3.0$. From these response curves, it is quite evident that with only three modes, Galerkin's solution becomes very close to the matrix exponential solution. Figure 3 compares the response curves for identical parameters but $t_f = 5.0$. 
Figure 1: A two degrees-of-freedom spring-mass-damper system.
Figure 2: The optimal response curves: Comparison of matrix exponential solution and Galerkin solution for $t_f = 3$ seconds with 4 modes.

Figure 3: The optimal response curves: Comparison of matrix exponential solution and Galerkin solution for $t_f = 5$ seconds with 5 modes.
2.5.2 A four degree-of-freedom system

A four degree-of-freedom spring-mass-damper system is used as an example. The system is sketched in Figure 4. It consists of eight state variables and four control inputs. The matrices $A$ and $B$ for this system are as follows:

$$
A = \begin{bmatrix}
-M^{-1}C & -M^{-1}K \\
I_4 & 0
\end{bmatrix}
$$

(2.37)
where the matrices $M$, $C$, and $K$ are

$$M = \begin{bmatrix} m_1 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 \\ 0 & 0 & m_3 & 0 \\ 0 & 0 & 0 & m_4 \end{bmatrix}$$

$$C = \begin{bmatrix} c_1 + c_2 & -c_2 & 0 & 0 \\ -c_2 & c_2 + c_3 & -c_3 & 0 \\ 0 & -c_3 & c_3 + c_4 & -c_4 \\ 0 & 0 & -c_4 & c_4 + c_5 \end{bmatrix}$$

$$K = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 \\ -k_2 & k_2 + k_3 & -k_3 & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 \\ 0 & 0 & -k_4 & k_4 + k_5 \end{bmatrix}$$

The parameters in MKS units are $m_1 = m_2 = m_3 = m_4 = 1.0$, $c_1 = c_3 = c_5 = 1.0$, $c_2 = c_4 = 2.0$, $k_1 = k_2 = k_3 = k_4 = k_5 = 3.0$. The weighted function $\overline{Q} = 0$ and $R$ is identity of a consistent dimension. The boundary conditions are
\[ q(t_0) = (10, 20, 10, 20)^T, \quad \bar{q}(t_0) = (10, 20, 10, 20)^T, \quad q(t_f) = (0, 0, 0, 0)^T \quad \text{and} \quad \bar{q}(t_f) = (0, 0, 0, 0)^T \]

Figure 5 compares the optimal response curves of the state variables \( q_1, q_2, q_3 \) and \( q_4 \) obtained using Matrix-exponential method and Galerkin's method for \( t_f = 1.0 \). From these response curves, it is quite evident that with only two modes, Galerkin's solution becomes very close to the matrix exponential solution.

![Figure 5: The optimal response curves: Comparison of matrix exponential solution and Galerkin solution for \( t_f = 1 \) seconds with 8 modes.](image-url)
2.6 Summary

In this chapter, a higher-order procedure was described to solve the optimization problem for a spring-mass-damper system which has as many inputs as the number of degrees of freedom. In this procedure, the optimality equations were written as $k$ 4th order differential equation as opposed to conventional procedure which result in $4k$ first order differential equations. The 4th order equations were solved using weighted residual procedure and the solutions were shown to be very close to the conventional procedure obtained by matrix exponential methods.
Chapter 3

Optimization Problems for the Case (n = pm)

3.1 Introduction

This chapter deals with optimization of a class of linear dynamic systems with \( n \) states and \( m \) control inputs, commanded to move between two fixed states in a prescribed final time. Also, a new procedure for dynamic optimization is presented that does not use Lagrange multipliers. The equations for linear dynamic systems can be rewritten in the following form:

\[
\dot{X} = A X + B U
\]  

(3.1)

where \( A \) is a \((n \times n)\) matrix, \( X \) is a \((n \times 1)\) state vector, \( B \) is a \((n \times m)\) matrix, and \( U \) is a \((m \times 1)\) control vector. The statement of the problem is to determine \( X(t) \) that takes the system from an initial state \( X_0 \) at time \( t_0 \) to a final state \( X_f \) at \( t_f \). The path during \( t_0 \) and \( t_f \) must minimize the quadratic functional

\[
J = \frac{1}{2} \int_{t_0}^{t_f} [\dot{X}^T \overline{Q} X + U^T R U] dt
\]  

(3.2)

where \( \overline{Q} \) is a \((n \times n)\) symmetric matrix and \( R \) is a \((m \times m)\) symmetric and positive definite matrix. The conventional way of handling this problem ([2], [6]) is to augment the state Eq. (3.1) to the cost functional \( J \) using Lagrange's multipliers. Then, using the principles of variational calculus, the necessary conditions for extremum are written. As described in chapter 1, necessary conditions are expressed as \( 2n \) first-order differential equations in the
state and costate variables. The boundary conditions are given state variables at \( t_0 \) and \( t_f \).

The resulting two-point boundary value problem (TPBVP) with \( 2n \) first-order differential equations can be solved by Matrix-exponential methods. The solution requires matrix inversion and other operations on \((n \times n)\) matrices.

In this Chapter, the state equations are transformed into a new form so that \( U \) and \( X \) can be solved explicitly in terms of higher-order derivatives of a subset of the state variables. On substituting the expressions for \( U \) and \( X \) in \( J \), the constraint equations can be completely eliminated and the cost functional can be written in terms of higher-order derivatives of this subset. Since the problem becomes unconstrained, the need for Lagrange multipliers is eliminated. On using the variational principles, the necessary conditions for optimality are derived which are \( m \) differential equations of order \( 2p \). The solution of these \( m \) equations requires \( 2pm \) boundary conditions which are obtained from the given boundary conditions of \( X \) at \( t_0 \) and \( t_f \). These equations are solved using the weighted residual techniques.

The organization of this paper is as follows: section 3.2 describes a special state transformation and the resulting state equations. Section 3.3 states the new cost functional and the necessary conditions for its optimality. The procedure for obtaining the solution using weighted residual technique is described in section 3.4. This procedure is illustrated by examples in section 3.5.

### 3.2 State Transformation
On applying a linear transformation $\bar{X} = TX$, $(\bar{A}, \bar{B})$ transforms to $(A, B)$ where $\bar{A}T = TA$ and $\bar{B} = TB$. In this study, a special form of $(A, B)$ is sought which allows explicit solution of $U$ and $X$ in terms of higher derivatives of a subset of the state variables. It can be easily verified that for these requirements, the transformation is not unique. In this chapter, we choose the matrix $T = [\bar{B} \bar{A} \bar{A}^2 \bar{B} \ldots \bar{A}^{p-1} \bar{B}]$, assuming that this matrix is nonsingular. With this choice, $A$ and $B$ matrices take the following form:

$$ A = \begin{bmatrix} 0 & 0 & 0 & \cdots & E_1 \\ I_m & 0 & 0 & \cdots & E_2 \\ 0 & I_m & 0 & \cdots & E_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_m & E_p \end{bmatrix}, \quad B = \begin{bmatrix} I_m \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (3.3) $$

where $E_i$ are $(m \times m)$ matrices, $I_m$ is an $m$-dimensional identity matrix, and $0$ are $(m \times m)$ zero matrices. It can be shown that

$$ \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_p \end{bmatrix} = T^{-1} \bar{A}^p \bar{B} \quad (3.4) $$

From this new form of $A$ and $B$, the control vector $U$ and the state vector $X$ can be written as higher derivatives of $X_p$, an $m$-dimensional vector consisting of the entries $[x_{(p-1)m+1}, x_{(p-1)m+2}, \ldots, x_{pm}]^T$ of the vector $X$. 
\[
\begin{bmatrix}
U \\
X_1 \\
\vdots \\
X_{p-2} \\
X_{p-1} \\
X_p
\end{bmatrix} =
\begin{bmatrix}
-E_1 & -E_2 & \cdots & -E_p & I_m \\
-E_2 & -E_3 & \cdots & I_m & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
-E_p & I_m & \cdots & 0 & 0 \\
I_m & 0 & \cdots & 0 & 0
\end{bmatrix}
\begin{bmatrix}
X_p \\
\dot{X}_p \\
\vdots \\
\dddot{X}_p \\
\dddot{X}_p
\end{bmatrix}
\]

(3.5)

and define

\[
C =
\begin{bmatrix}
-E_1 & -E_2 & \cdots & -E_p & I_m \\
-E_2 & -E_3 & \cdots & I_m & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
-E_p & I_m & \cdots & 0 & 0 \\
I_m & 0 & \cdots & 0 & 0
\end{bmatrix}
\]

(3.6)

From the given boundary conditions of \(\bar{X}\) at \(t_0\) and \(t_f\), the boundary conditions on the higher derivatives \(X_p, \dot{X}_p, \dddot{X}_p, \ldots, \dddot{X}_p\) can be computed from Eq.(3.6) at \(t_0\) and \(t_f\).

### 3.3 Optimality Conditions

On substituting \(\bar{X} = TX\) in Eq.(3.2), the cost functional \(J\) becomes:

\[
J = \frac{1}{2} \int_{t_0}^{t_f} \left[U^T \ X^T \ W \ [U \ X] \right] dt
\]

(3.7)

where

\[
W = \begin{bmatrix}
R & 0 \\
0 & T^T \bar{Q} T
\end{bmatrix}
\]

(3.8)

On substituting Eq.(3.5) in Eq.(3.7), the cost functional \(J\) can be written as
\[ J = \int_{t_0}^{t_f} F(X_p, \dot{X}_p, \ddot{X}_p, \ldots, \dddot{X}_p) \, dt \quad (3.9) \]

where

\[
F = \frac{1}{2} \begin{bmatrix} X_p & \dot{X}_p & \ldots & \dddot{X}_p \end{bmatrix} \begin{bmatrix} C^T \omega C \end{bmatrix} \begin{bmatrix} X_p \\ \dot{X}_p \\ \vdots \\ \dddot{X}_p \\ X_p \end{bmatrix} \quad (3.10)
\]

From the principles of variational calculus,

\[
\delta J = \int_{t_0}^{t_f} \left( \frac{\partial F}{\partial X_p} \frac{d}{dt} - \frac{\partial F}{\partial \dot{X}_p} \frac{d}{dt} \right) + \ldots + \left( \frac{\partial F}{\partial \dddot{X}_p} \frac{d^4}{dt^4} \right) \, dt
\]

\[
+ h^T \left[ \frac{\partial F}{\partial \dot{X}_p} \frac{d}{dt} - \frac{\partial F}{\partial \dot{X}_p} \frac{d}{dt} \right] + \ldots + \left( \frac{\partial F}{\partial \dddot{X}_p} \frac{d^4}{dt^4} \right) \, \bigg|_{t_0}^{t_f}
\]

\[
+ \ldots + h^T \left[ \frac{\partial F}{\partial \dddot{X}_p} \right] \bigg|_{t_0}^{t_f}
\]

where \( h = \delta X_p, \dot{h} = \delta \dot{X}_p, \) etc. are respectively the variation of \( X_p \) and its higher derivatives. From section 3.2, the boundary values of \( X_p, \dot{X}_p, \ddot{X}_p, \ldots, \dddot{X}_p \) at \( t_0 \) and \( t_f \) are known. As a result, \( h, \dot{h}, \dddot{h}, \ldots, \dddot{h} \) at \( t_0 \) and \( t_f \) are zero. Hence, the necessary condition for optimality for fixed states at \( t_0 \) and \( t_f \) becomes

\[
\frac{\partial F}{\partial X_p} \frac{d}{dt} - \frac{\partial F}{\partial \dot{X}_p} \frac{d}{dt} + \ldots + (-1)^p \frac{d^p}{dt^p} \frac{\partial F}{\partial \dddot{X}_p} = 0
\]

(3.12)
Since $F$ has a symmetric quadratic form,

$$
\begin{bmatrix}
\frac{\partial F}{\partial X_p} \\
\frac{\partial F}{\partial \dot{X}_p} \\
\vdots \\
\frac{\partial F}{\partial \dddot{X}_p}
\end{bmatrix}
= [C^T WC]
\begin{bmatrix}
X_p \\
\dddot{X}_p \\
\vdots \\
\dddot{X}_p
\end{bmatrix}
$$

(3.13)

On substituting the expressions for partial derivatives from Eq.(3.13) in Eq.(3.12), the necessary conditions can be written as

$$
\sum_{f=0}^{2p} S_f \dddot{X}_p = 0
$$

(3.14)

where $S_f$ is a $(m \times m)$ matrix. The highest order of the derivative in the optimality equation is $2p$. The boundary conditions are the specified vectors $X_p, \dot{X}_p, \dddot{X}_p, ..., \dddot{X}_p^{-1}$ at $t_0$ and $t_f$. In the special case of a single input, Eq.(3.14) reduces to a single equation having the highest derivative as $2n$.

### 3.4 Weighted Residual Solution

Different forms of weighted residual methods have been used to solve boundary value problems. A summary of these methods is available [1]. Due to the nature of the optimization problem with fixed end constraints at $t_0$ and $t_f$, only those methods are considered suitable that satisfy the differential equations approximately over the domain.
but satisfy the boundary conditions exactly such as in the Galerkin's method, method of
moments, and collocation method. In this Chapter, the method of moments was selected
for the solution of the problem since it is computationally simpler than Galerkin's method.
The mode functions are chosen as polynomials. The admissible form of the solution is
selected as [3]:

\[ X_p(t) = \Phi_0(t) + \sum_{i=1}^{k} L_i \phi_i(t) \]  

(3.15)

where \( \Phi_0(t) \) is an m-dimensional vector of mode functions that satisfies the boundary
conditions of the vector \( X_p \) at time \( t_0 \) and \( t_f \) exactly. \( \phi_i(t) \) are mode functions chosen to
have the derivatives up to order \( p-1 \) zero at two end points and \( L_1, \ldots, L_k \) are m-
dimensional unknown vectors to be determined. Under these conditions, \( X_p(t) \) satisfies
the boundary conditions of the problem regardless of the numerical values of \( L_1, \ldots, L_k \).

On substituting Eq.(3.15) in Eq.(3.14), the following error vector results:

\[ \varepsilon(t) = \sum_{j=0}^{2p} S_j [\Phi_0(t) + \sum_{i=1}^{k} L_i \phi_i(t)] \]  

(3.16)

which is made to be orthogonal to the polynomial basis functions \( \psi_1 = 1, \psi_2 = t, \)
\( \psi_3 = t^2, \ldots, \psi_k = t^{k-1} \):

\[ \int_{0}^{t} \varepsilon(t) \psi_i(t) dt = 0, \quad i = 1, \ldots, k \]  

(3.17)

This leads to \( mk \) scalar equations which can be used to solve for the \( mk \) elements of the
vector \( L_1, \ldots, L_k \). The equations (3.17) can be written in a block matrix form:
\[ \begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1k} \\ T_{21} & T_{22} & \cdots & T_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ T_{k1} & T_{k2} & \cdots & T_{kk} \end{bmatrix} \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_k \end{bmatrix} = \begin{bmatrix} -R_1 \\ -R_2 \\ \vdots \\ -R_k \end{bmatrix} \] (3.18)

where \( T_{q1} \) is a \((m \times m)\) matrix subblock, and \( R_q \) is a \((m \times 1)\) vector defined below:

\[
T_{q1} = \int_{t_0}^{t_f} \sum_{j=0}^{2p} S_j \phi_i(t) \psi_q dt
\] (3.19)

\[
R_q = \int_{t_0}^{t_f} \sum_{j=0}^{2p} S_j \Phi_0(t) \psi_q dt
\]

The above equation is inverted to solve for the vectors \( L_1, \ldots, L_k \).

### 3.4.1 Mode Functions

It is evident that any set of \( \Phi_0(t) \) and \( \phi_i(t) \) that satisfies the selection criteria is valid. In this Chapter, \( \Phi_0(t) \) is chosen to be the following polynomial function of time

\[
\Phi_0(t) = \sum_{i=0}^{p} F_i t^i + \sum_{i=1}^{p-1} F_{pi} t^i (t - t_f)^i
\] (3.20)

where \( t_0 = 0 \). The coefficients \( F_i \) are vectors of dimension \( m \) which are solved using the given boundary conditions of \( X_p \) and its derivatives at time \( t_0 \) and \( t_f \). This choice of \( \Phi_0(t) \) results in a special structure in the computation of \( F_0, F_1, \ldots, F_{2p-1} \).

In order to bring out this special structure, we focus on the \( i \)th component of \( \Phi_0(t) \). The relevant coefficients are \( F_{0i}, F_{1i}, \ldots, F_{2p-1i} \) which must be determined using the boundary values of the \( i \)th variable \( x_i \) of \( X_p \). It can be shown that \( F_{0i}, F_{1i}, \ldots, F_{2p-1i} \) satisfy the following equations:
where \( \Lambda, U, \) and \( L \) are respectively diagonal, upper triangular, and lower triangular square matrices of dimension \( p \). The expressions for \( \Lambda, U, \) and \( L \) are given in Appendix A Eq.(A4). Also, \( \Lambda, U, \) and \( L \) stay the same for every variable \( i \). The vectors \( V_{ui}, V_{2i}, W_{ui}, \) and \( W_{2i} \) are defined below:

\[
V_{ui} = \begin{bmatrix} F_{0i} \\ F_{1i} \\ \vdots \\ F_{p-1i} \end{bmatrix}, \quad V_{2i} = \begin{bmatrix} F_{p} \\ F_{p+1} \\ \vdots \\ F_{2p-1} \end{bmatrix}
\]

\[
W_{ui} = \begin{bmatrix} x_{i0} \\ \vdots \\ x_{i0} \\ \vdots \\ \vdots \\ x_{i0} \end{bmatrix}, \quad W_{2i} = \begin{bmatrix} x_{if} \\ \vdots \\ x_{if} \\ \vdots \\ \vdots \\ x_{if} \end{bmatrix}
\]

where \( x_i \) is the \( i \)th element of \( X_p \). From Eq.(3.21), \( \Lambda V_{ui} = W_{ui} \) and \( LV_{2i} = W_{2i} - UV_{ui} \).

With the special structures of \( \Lambda \) and \( L \), \( V_{ui} \) and \( V_{2i} \) can be computed without matrix inversion. The mode functions \( \phi_i(t) \) are selected as

\[
\phi_i(t) = t^{p-1+i}(t - t_f)^p, \quad i = 1, \ldots, k
\]

These mode functions possess the property of having zero derivatives up to the order \( p-1 \) at time \( t_0 \) and \( t_f \). For these mode functions, the expressions for matrices \( T_{pl} \) and the right hand side vectors \( R_p \) are listed in Appendix A Eq.(A5) and Eq.(A6).

### 3.4.2 Recursion with modes
The matrix and vectors of Eq.(3.18) possess certain characteristics which allow the computation of $L_1, \ldots, L_k$ in a recursive way as the number of modes is increased. Let $\tau_{k+1}$ be the left-hand side matrix of Eq.(3.18) for $k+1$ mode functions:

$$
\tau_{k+1} = \begin{bmatrix}
T_{11} & T_{12} & \cdots & T_{1k} & T_{1k+1} \\
T_{21} & T_{22} & \cdots & T_{2k} & T_{2k+1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
T_{k1} & T_{k2} & \cdots & T_{kk} & T_{kk+1} \\
T_{k+11} & T_{k+12} & \cdots & T_{k+1k} & T_{k+1k+1}
\end{bmatrix}
$$

(3.25)

It can be shown that $\tau_{k+1}$ is related to $\tau_k$ for $k$ modes in the following way:

$$
\tau_{k+1} = \begin{bmatrix}
\tau_k & \tau_{kk+1} \\
\tau_{k+1k} & \tau_{k+1k+1}
\end{bmatrix}
$$

(3.26)

where $\tau_{kk+1}$ is the column block and $\tau_{k+1k}$ is the row block of $\tau_{k+1}$. This happens because with increase in number of modes from $k$ to $k+1$, the first $(k \times k)$ block within $\tau_{k+1}$ is same as $\tau_k$. Let us denote the solution vector with $k+1$ modes as

$$
\ell_{k+1} = \begin{bmatrix}
L_1 \\
L_2 \\
\vdots \\
L_k \\
L_{k+1}
\end{bmatrix}
$$

(3.27)

This solution vector $\ell_{k+1}$ can be written from $\ell_k$ in the following incremental way

$$
\ell_{k+1} = \begin{bmatrix}
\ell_k + \delta \ell_k \\
L_{k+1}
\end{bmatrix}
$$

(3.28)

Similarly, the right-hand side vector $R_{k+1}$ for $k+1$ modes is written as
which can be written from $\mathcal{R}_k$ in the following incremental way

$$\mathcal{R}_{k+1} = \begin{bmatrix} -R_1 \\ -R_2 \\ \vdots \\ -R_k \\ -R_{k+1} \end{bmatrix}$$ (3.29)

Using matrix properties [6], it can be shown that

$$\mathcal{R}_{k+1} = \begin{bmatrix} \mathcal{R}_k \\ -R_{k+1} \end{bmatrix}$$ (3.30)

Using matrix properties [6], it can be shown that

$$[\tau_{k+1}]^{-1} = \begin{bmatrix} [\tau_k]^{-1} + \Delta_2 \Delta_1^{-1} \Delta_3 & -\Delta_2 \Delta_1^{-1} \\ -\Delta_1^{-1} \Delta_3 & \Delta_1^{-1} \end{bmatrix}$$ (3.31)

where

$$\Delta_1 = T_{k+1k+1} - \tau_{k+1k} [\tau_k]^{-1} \tau_{kk+1}$$
$$\Delta_2 = [\tau_k]^{-1} \tau_{kk+1}$$
$$\Delta_3 = \tau_{k+1k} [\tau_k]^{-1}$$ (3.32)

From here, it can be verified that

$$\delta \ell_k = \Delta_3 \Delta_1^{-1} \left[ \Delta_3 \mathcal{R}_k + R_{k+1} \right]$$
$$L_{k+1} = -\Delta_1^{-1} \left[ \Delta_3 \mathcal{R}_k + R_{k+1} \right]$$ (3.33)

Eq.(3.33) states that with an increase in the number of modes from $k$ to $k+1$, the solution $\ell_{k+1}$ can be obtained from $\ell_k$ in an incremental way. This incremental computation requires inversion of a $(m \times m)$ matrix instead of inverting the full $[m(k+1) \times m(k+1)]$ matrix.

In summary, in order to compute $\ell_{k+1}$ for $k+1$ modes, the following procedure can be adopted: (i) From the solution with $k$ modes $\ell_k$, $\tau_k^{-1}$, $\mathcal{R}_k$ are available, (ii) From the
information of $k+1$th mode, form the matrices $\tau_{kk+1}$, $\tau_{k+1k}$ and $\tau_{k+1k+1}$, (iii) compute $\delta \ell_k$ and $L_{k+1}$ from Eq.(3.33) and form $\ell_{k+1}$.

3.4.3 Overall Computation

Figure 6: A computational flowchart for the higher-order algorithm
The steps of the computation with the higher-order algorithm are outlined in a flowchart in Figure 6. From the perspective of real time trajectory generation, the computations can be divided into (i) on-line, and (ii) off-line. The on-line computations depend on parameters of the desired motion, e.g., \( t_f \) and \( X_f \) and must be done after the issuance of the command. The results of the off-line computations, on the other hand, may be available as stored data. Among the steps outlined in the flowchart, \( A, B, T, T^{-1} \) and the matrices \( S_0, S_1, \ldots, S_{2p-1} \) of the higher order necessary conditions can be computed off-line from the state-space model \((\overline{A}, \overline{B})\) of the original system. The remaining steps of the flowchart must be performed on-line.

From the flowchart, we observe that in the steps leading to the computation of \( X_p(t) \), except for state transformation between \( \overline{X}(t) \) and \( X(t) \), matrix inversion, matrix multiplication, and vector multiplication are needed only for \((m \times m)\) matrices. In a special case, where \( m = 1 \), these computations reduce to scalar computations. It is well known ([2], [6]) that matrix exponential solution of the necessary conditions derived with Lagrange's multipliers requires matrix inversion, matrix multiplication, and vector multiplication of \((n \times n)\) matrices, where \( n \) is the number of state variables. As a result, the weighted residual solution of the higher-order necessary conditions offers a distinct computational advantage over the matrix exponential solution of the necessary conditions derived using Lagrange's multipliers. This distinction becomes more and more pronounced as the ratio \( p \) between the number of states and control inputs increase, due to two reasons: (i) the significant matrix computations are of the order \( m \), (ii) it is no longer
necessary to select a large number of modes \( k \) because boundary values of \( X_p \), up to \((p-1)\)th derivative, at time \( t_0 \) and \( t_f \) are exactly known. In practice, these boundary values are enough to give a very good approximation to the exact solution.

![Diagram of a spring-mass-damper system with eight states and two inputs.](image)

Figure 7: A spring-mass-damper system with eight states and two inputs.

### 3.5 Illustrative Example

A four degree-of-freedom spring-mass-damper system is used as an example. The system is sketched in figure 7. It consists of eight state variables and two control inputs. The matrices \( \bar{A} \) and \( \bar{B} \) for this system are as follow:
\[ \bar{A} = \begin{bmatrix} 0 & I_4 \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \]

\[ \bar{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{m_1} \\ \frac{1}{m_2} \\ \frac{1}{m_3} \end{bmatrix} \]  \hspace{1cm} (3.34)

where the matrices \( M, C, \) and \( K \) are

\[ M = \begin{bmatrix} m_1 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 \\ 0 & 0 & m_3 & 0 \\ 0 & 0 & 0 & m_4 \end{bmatrix} \]

\[ C = \begin{bmatrix} c_1 + c_2 & -c_2 & 0 & 0 \\ -c_2 & c_2 + c_3 & -c_3 & 0 \\ 0 & -c_3 & c_3 + c_4 & -c_4 \\ 0 & 0 & -c_4 & c_4 + c_5 \end{bmatrix} \]

\[ K = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 \\ -k_2 & k_2 + k_3 & -k_3 & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 \\ 0 & 0 & -k_4 & k_4 + k_5 \end{bmatrix} \]  \hspace{1cm} (3.35)

The parameters in MKS units are \( m_1 = m_2 = m_3 = m_4 = 1.0, \ c_1 = c_3 = c_5 = 1.0, \ c_2 = c_4 = 2.0, \ k_1 = k_2 = k_3 = k_4 = k_5 = 3.0. \) The weighted function \( \bar{Q} = 0 \) and \( R \) is identity of a consistent dimension. The boundary conditions are

\[ \bar{X}(0) = (10, 20, 10, 20, 10, 20, 10, 20)^T \] and \( \bar{X}(t_f) = (0, 0, 0, 0, 0, 0, 0, 0)^T. \)
Figure 8: The optimal trajectories for a single input $u_1$ and final time $t_f = 1$ second. The solutions from zero mode, one mode, and matrix exponential method overlap within the accuracy of drawing.

Figure 9: The optimal trajectories for a single input $u_1$ and $u_3$ and final time $t_f = 1$ second. The solutions from zero mode, one mode, and matrix exponential method overlap within the accuracy of drawing.
The results of the optimization for (i) a single input $u_1$, (ii) two inputs $u_1$ and $u_3$ are plotted in the figure 8 and 9 along with the respective solutions using matrix exponential method with Lagrange multipliers. We observe from these plots that the solution with zero mode is itself a good approximation to the exact solution.

3.6 Summary

A new higher-order two-points boundary value formulation has been presented for optimization of a class of linear dynamic systems. For a system with $n$ states and $m$ control inputs, the higher-order formulation requires solution of $m$ $2p$-order differential equations. In contrast, the conventional approach solves the same problem using $2n$ first order differential equations in the state and costate variables. The new solution approach presented in this Chapter requires matrix operations, primarily, on $(m \times m)$ matrices while the conventional approach requires operation on $(n \times n)$ matrices. The new solution approach has distinct computational advantages for problems where $p$, the ratio between $n$ and $m$ is large. The reasons are: (i) matrix operations are carried out on $(m \times m)$ matrices instead of $(n \times n)$ matrices, (ii) the knowledge of $2p$ boundary values (up to $p$-1th derivative at the two end time) results in a reasonable approximation of the solution. As a result, the solution is achieved with a very small number of modes. Similar procedures are currently being applied to both time-varying linear and nonlinear dynamic systems.
Chapter 4

Optimization Problem for the Case \(( n = rm )\)

4.1 Introduction

This chapter deals with optimization of a class of linear dynamic systems with \( n \) states and \( m \) control inputs, commanded to move between two fixed states in a prescribed final time. Also, a new procedure for dynamic optimization is presented that does not use Lagrange multipliers. The equations for linear dynamic systems can be written in the following form:

\[
\dot{X} = \bar{A}X + \bar{B}U
\]  

(4.1)

where \( \bar{A} \) is a \((n \times n)\) matrix, \( \bar{X} \) is a \((n \times 1)\) state vector, \( \bar{B} \) is a \((n \times m)\) matrix, and \( U \) is a \((m \times 1)\) control vector. The statement of the problem is to determine \( X(t) \) that takes the system from an initial state \( \bar{X}_0 \) at time \( t_0 \) to a final state \( \bar{X}_f \) at \( t_f \). The path during \( t_0 \) and \( t_f \) must minimize the quadratic functional

\[
J = \frac{1}{2} \int_{t_0}^{t_f} (\bar{X}^T Q \bar{X} + U^T R U) dt
\]  

(4.2)

where \( Q \) is a \((n \times n)\) symmetric matrix and \( R \) is a \((m \times m)\) symmetric and positive definite matrix.

In the chapter 3, we proposed a higher-order two-point boundary value formulation for optimization of a class of dynamic systems with \( n = pm \), where \( p \) is an integer. There, it was demonstrated that a higher-order formulation solved by weighted
residual method is computationally more efficient than conventional methods. This chapter generalizes these results to arbitrary $n$ and $m$. The organization of this chapter is as follows: section 4.2 describes a state transformation so that the entire state and control vector can be written in terms of higher derivatives of a subset of the states. section 4.3 describes the necessary conditions for optimality of the resulting unconstrained cost functional. A numerical procedure for solve the optimality equations is described in section 4.4. Some examples illustrating the method are presented in section 4.5.

4.2 State Transformation

On applying a linear transformation $\bar{X} = TX$, $(\bar{A}, \bar{B})$ transforms to $(A, B)$ where $\bar{A} = TA$ and $\bar{B} = TB$. If $T$ is selected to be a nonsingular matrix of the following form:

$$T = [\bar{B}_1 \bar{A} \bar{B}_1 \ldots \bar{A}^{n-1} \bar{B}_1 | \bar{B}_2 \bar{A} \bar{B}_2 \ldots \bar{A}^{n-1} \bar{B}_2 | \ldots | \bar{B}_p \bar{A} \bar{B}_p \ldots \bar{A}^{n-1} \bar{B}_p ],$$

$$p \leq m \tag{4.3}$$

the state equation of Fig. 10 can be broken up into $p$ component blocks where the first block has the following structure:

$$\begin{bmatrix}
\vdots \\
x_2 \\
x_3 \\
\vdots \\
x_{n_1} \\
\vdots \\
x_{n_1+1-1}
\end{bmatrix}
= \begin{bmatrix}
x_1 \\
\vdots \\
x_2 \\
\vdots \\
x_{n_1} \\
\vdots \\
x_{n_1+1-1}
\end{bmatrix}
+ \begin{bmatrix}
A_{21} & A_{22} & \cdots & A_{2n_2} & \cdots & A_{2n_2+1-n_2} & \cdots & A_{2n_2+1-n_2} \\
A_{31} & A_{32} & \cdots & A_{3n_3} & \cdots & A_{3n_3+1-n_3} & \cdots & A_{3n_3+1-n_3} \\
\vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \cdots & \vdots \\
A_{n_11} & A_{n_12} & \cdots & A_{n_1n_1} & \cdots & A_{n_1n_1+1-n_1} & \cdots & A_{n_1n_1+1-n_1} \\
\end{bmatrix}
\begin{bmatrix}
x_{n_1} \\
x_{n_1+1} \\
x_{n_1+2} \\
\vdots \\
x_{n_1+n_2} \\
x_{n_1+n_2+1} \\
x_{n_1+n_2+2} \\
\vdots \\
x_{n_1+n_2+n_p}
\end{bmatrix} \tag{4.4}$$
Figure 10: The structure of resulting state equation in $\bar{X}$ with the transformation $\bar{X} = TX$, where $T$ is described in equation (4.3)
Along the same pattern, the subsequent blocks, \( i = 2, \ldots, p \), are:

\[
\begin{bmatrix}
    x_{i1} + \cdots + x_{i1} + 2 \\
    \vdots \\
    x_{i1} + \cdots + x_{i1} + p \\
\end{bmatrix} = \begin{bmatrix}
    x_{i1} + \cdots + x_{i1} + 1 \\
    \vdots \\
    x_{i1} + \cdots + x_{i1} + p \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
    A_{n1+\cdots+n1+2} & A_{n1+\cdots+n1+2+n2} & \cdots & A_{n1+\cdots+n1+2+n2+n3} \\
    A_{n1+\cdots+n1+3} & A_{n1+\cdots+n1+3+n2} & \cdots & A_{n1+\cdots+n1+3+n2+n3} \\
    \vdots & \vdots & \ddots & \vdots \\
    A_{n1+\cdots+n1} & A_{n1+\cdots+n1+n2} & \cdots & A_{n1+\cdots+n1+n2+n3} \\
\end{bmatrix}
\begin{bmatrix}
    x_{n1} \\
    x_{n1+n2} \\
    \vdots \\
    x_{n1+\cdots+n1+p} \\
\end{bmatrix}
\]

\( (i = 2, \ldots, p) \)

In Fig. 10, the rows that contain the control inputs \( u_1, u_2, \ldots, u_p \) can be combined together into the following block:

\[
\begin{bmatrix}
    x_1 \\
    x_{n1+1} \\
    \vdots \\
    x_{n1+\cdots+n1+p+1} \\
\end{bmatrix} = \begin{bmatrix}
    u_1 \\
    u_2 \\
    \vdots \\
    u_p \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
    A_{1n1} & A_{1n1+n2} & \cdots & A_{1n1+n2+n3} \\
    \frac{A}{n1+n2} & \frac{A}{n1+n2+n3} & \cdots & \frac{A}{n1+n2+n3+n4} \\
    \vdots & \vdots & \ddots & \vdots \\
    A_{n1+\cdots+n1+1} & A_{n1+\cdots+n1+1+n2} & \cdots & A_{n1+\cdots+n1+1+n2+n3} \\
\end{bmatrix}
\begin{bmatrix}
    x_{n1} \\
    x_{n1+n2} \\
    \vdots \\
    x_{n1+\cdots+n1+p} \\
\end{bmatrix}
\]

\( (4.6) \)

### 4.2.1 State Variables as Higher Derivatives

From the first component equation of Eq.(4.6) and the block Eq.(4.4), the variables \( u_1, x_1, x_2, \ldots, x_n \) can be written in terms of higher derivatives of the variables \( x_{n1}, x_{n1+n2}, \ldots, x_{n1+n2+\cdots+n1+p} \). Their expressions are:
where the symbol $r_i$ represents the $i$th row of the matrix formed by stacking the column vectors $A_1, A_{n_1}, \ldots, A_{n_1+n_2}, \ldots, A_{n_1+n_2+\ldots+n_p}$. $e_1$ is a unit row vector with zero entries except for 1 in the first element. A symbol $x_{n_a}$ denotes $i$th derivative of variable $x_{n_a}$. The above equation can be written compactly in a matrix form:

$$X_1 = \sum_{j=0}^{n} C_{lj} X_s$$  \hspace{0.5cm} (4.8)$$

where $X_1 = (u_1, x_1, x_2, \ldots, x_{n-1}, x_n)^T$ and $X_s = (x_{n_1}, x_{n_1+n_2}, \ldots, x_{n_1+n_2+\ldots+n_p})^T$. Similarly, $u_1, x_{n_1+n_2+\ldots+n_p+1}, x_{n_1+n_2+\ldots+n_p+2}, \ldots, x_{n_1+n_2+\ldots+n_p}, x_{n_1+n_2+\ldots+n_p}$ can be written in terms of higher derivatives of $x_{n_1}, x_{n_1+n_2}, \ldots, x_{n_1+n_2+\ldots+n_p}$. These expression are:
where the symbol \( r_{n_1+\ldots+n_{i-1}+3} \) denotes the \((n_1+\ldots+n_{i-1}+3)\)th row of the matrix formed by stacking the column vectors \( A_{n_1}, A_{n_1+n_2}, \ldots, A_{n_1+n_2+\ldots+n_p} \). \( e_i \) is a unit row vector with zero entries except for 1 in the \(i\)th place. The symbol \( x_{n_i} \) denotes \(n_i\)th derivative of the variable \( x_{n_i} \). The above equation can be written compactly in a matrix form:

\[
X_i = \sum_{j=0}^{n_i} C_{ij} \dot{X}_s, \quad (\forall i = 2, \ldots, p)
\]

(4.10)

where \( X_i = (u_i, x_{n_1+\ldots+n_{i-1}+1}, x_{n_1+n_2+\ldots+n_{i-1}+2}, \ldots, x_{n_1+\ldots+n_{i-1}}, x_{n_1+\ldots+n_p})^T \). The vectors \( X_1, X_2, \ldots, X_p \) can be stacked below each other to result in the following form:

\[
\begin{bmatrix}
X_1 \\
X_2 \\
\vdots \\
X_p
\end{bmatrix} =
\begin{bmatrix}
C_{10} & C_{12} & \cdots & C_{1N} \\
C_{20} & C_{22} & \cdots & C_{2N} \\
\vdots & \vdots & \ddots & \vdots \\
C_{p0} & C_{p2} & \cdots & C_{pN}
\end{bmatrix}
\begin{bmatrix}
\dot{X}_s \\
\ddots \\
\ddots \\
\ddots
\end{bmatrix}
\]

(4.11)

where \( N = \max(n_1, n_2, \ldots, n_p) \) and \( C_{\frac{i}{n_i+1}}, \ldots, C_{\frac{i}{n_i+1}} \) are zeros if \( n_i < N \).

### 4.2.2 Boundary Conditions

In this section, we look into the issue of computing boundary values of derivatives of \( X_s \) from the given boundary value data. Let us look at the structure of Eq.(4.7). We
observe that boundary values of \( \dot{x}_n \) can be computed from boundary values of \( x_{n-1} \) and \( x_n \). The boundary values of \( \ddot{x}_n \) requires boundary values of \( x_{n-2}, \dot{x}_n, x_{n+1}, \ldots, \dot{x}_{n+n_p} \). This pattern repeats in the computation of boundary values of higher derivatives of \( x_n \).

Similarly, there is a pattern in computation of higher derivatives of the variables \( x_{n_1+n_2+...+n_p} \). From Eq.(4.9), it can be shown that the general expression for computation of \( x_i \) in terms of its lower derivatives and elements of \( X \) is:

\[
\begin{bmatrix}
\vdots \\
x_{n_1} \\
\vdots \\
x_{n+n_2+...+n_p}
\end{bmatrix} = \begin{bmatrix}
\vdots \\
x_{n_1-i} \\
\vdots \\
x_{n+n_2+...+n_p-i}
\end{bmatrix} + \begin{bmatrix}
r_{n-i+1} \\
r_{n+i+1} \\
\vdots \\
r_{n+n_2+...+n_{i}+n_p-i+1}
\end{bmatrix} \begin{bmatrix}
X_{n_1} \\
X_{n+n_2} \\
\vdots \\
X_{n+n_2+...+n_p}
\end{bmatrix}
\]

\[
\begin{bmatrix}
r_{n-i+1} \\
r_{n+i+1} \\
\vdots \\
r_{n+n_2+...+n_{i}+n_p-i+1}
\end{bmatrix} \begin{bmatrix}
x_{n_1} \\
x_{n+n_2} \\
\vdots \\
x_{n+n_2+...+n_p}
\end{bmatrix} + \ldots + \begin{bmatrix}
r_{n-i+1} \\
r_{n+i+1} \\
\vdots \\
r_{n+n_2+...+n_{i}+n_p-i+1}
\end{bmatrix} \begin{bmatrix}
x_{n_1} \\
x_{n+n_2} \\
\vdots \\
x_{n+n_2+...+n_p}
\end{bmatrix}
\]

(4.12)

for \( i \leq N_m = \min(n_1 - 1, n_2 - 1, \ldots, n_p - 1) \).

If \( n_1 = n_2 = \ldots = n_p = N \), \( N_m = N - 1 \) and the boundary values of \( X_s, \dot{X}_s, \ddot{X}_s, \ldots, \dddot{X}_s \) can be computed from boundary values of \( X \) by recursively using Eq.(4.12). In case, \( n_1, n_2, \ldots, n_p \) are different from each other, not all boundary values of \( X_s \) up to (N-1)th derivative can be computed.
4.3 Optimization

On substituting $\overline{X} = TX$ in Eq.(4.2), the cost functional $J$ becomes:

$$J = \frac{1}{2} \int_{t_0}^{t_f} [X^T Q' X + U^T RU] dt$$  \hspace{1cm} (4.13)$$

where $Q' = T^T Q T$ is a $(n \times n)$ symmetric matrix. On using the definitions of $X_1, X_2, \ldots, X_p$ from the last section, $J$ can be rewritten in the following symmetric quadratic form:

$$J = \frac{1}{2} \int_{t_0}^{t_f} [X_1^T \quad X_2^T \quad \ldots \quad X_p^T] \begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1p} \\ Q_{21} & Q_{22} & \cdots & Q_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{p1} & Q_{p2} & \cdots & Q_{pp} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix} dt$$  \hspace{1cm} (4.14)$$

where $Q_y = \begin{bmatrix} R_y & 0 \\ 0 & Q_y' \end{bmatrix}$

where $R_y$ is a scalar and $Q_y'$ is a $(n_i + n_j)$ matrix. On substituting Eq.(4.11) in Eq.(4.14), the cost functional $J$ is expressed in terms of higher-order derivatives of the independent variables $X_s$:

$$J = \frac{1}{2} \int_{t_0}^{t_f} (X_s^T \quad \dot{X}_s^T \quad \ddot{X}_s^T \quad \ldots \quad N_s^T) C^T Q C \begin{bmatrix} X_s \\ \dot{X}_s \\ \ddot{X}_s \\ \vdots \\ N_s \end{bmatrix} dt$$  \hspace{1cm} (4.15)$$

where $C$ and $Q$ are respectively the large matrices of Eq.(4.11) and Eq.(4.14). It must be noted that Eq.(4.15) is a restatement of Eq.(4.1) and Eq.(4.2). An appropriate transformation matrix $T$ has been used to reduce the constrained optimization problem to an unconstrained optimization problem.
4.3.1 Variational Statement

From the principles of variational calculus, for fixed terminal time $t_0$ to $t_f$, the variation $\delta J$ of a functional $J$ of the form:

$$J = \int_{t_0}^{t_f} F(X_s, \dot{X}_s, \ddot{X}_s, \ldots, X_{s}^{\ldots, N})dt$$ (4.16)

is

$$\delta J = \int_{t_0}^{t_f} h^{T} \left[ \frac{\partial F}{\partial X_s} - \frac{d}{dt} \frac{\partial F}{\partial \dot{X}_s} + \ldots + (-1)^{N-1} \frac{d^{N-1}}{dt^{N-1}} \frac{\partial F}{\partial X_{s}^{N}} \right] dt$$

$$+ h^{T} \left[ \frac{\partial F}{\partial \dot{X}_s} - \frac{d}{dt} \frac{\partial F}{\partial \ddot{X}_s} + \ldots + (-1)^{N-2} \frac{d^{N-2}}{dt^{N-2}} \frac{\partial F}{\partial X_{s}^{N-1}} \right] \Big|_{t_0}^{t_f}$$

$$+ \ldots + h^{T} \left[ \frac{\partial F}{\partial X_{s}^{N-1}} \right] \Big|_{t_0}^{t_f}$$ (4.17)

where $X_s = (x_{s1}, x_{s2}, \ldots, x_{sp})^{T}$, $\frac{\partial F}{\partial X_s} = \left( \frac{\partial F}{\partial x_{s1}}, \frac{\partial F}{\partial x_{s2}}, \ldots, \frac{\partial F}{\partial x_{sp}} \right)^{T}$, and the variation $h = \delta X_s, \dot{h} = \delta \dot{X}_s, \ddot{h} = \delta \ddot{X}_s, \ldots, h = \delta X_{s}^{N-1}$. $\delta J$ consists of an integral term and several boundary terms. The necessary condition for extremum is that $\delta J$ must vanish.

4.3.2 Differential Equation

The integral in $\delta J$ vanishes if

$$\frac{\partial F}{\partial X_s} - \frac{d}{dt} \frac{\partial F}{\partial \dot{X}_s} + \ldots + (-1)^N \frac{d^N}{dt^N} \frac{\partial F}{\partial X_{s}^{N}} = 0$$ (4.18)
Due to symmetric and quadratic form of $F$ in Eq.(4.15),

$$
\begin{bmatrix}
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial \dot{X}_s} \\
\vdots \\
\frac{\partial F}{\partial \dddot{X}_s}
\end{bmatrix}
= C^T QC
\begin{bmatrix}
X_s \\
\dot{X}_s \\
\vdots \\
\dddot{X}_s
\end{bmatrix}
$$

(4.19)

On substituting the expressions for partial derivatives in Eq.(4.18), the optimality condition can be written in the following form:

$$
\sum_{j=0}^{2N} S_j \dddot{X}_s = 0
$$

(4.20)

where $S_j$ is a matrix of dimension $(p \times p)$ that multiplies $\dddot{X}_s$. The highest derivative of $X_s$ in the optimality equation is $2N$.

### 4.3.3 Boundary Terms

The boundary conditions required in the solution of Eq.(4.20) are obtained from the expression of $\delta J$ in Eq.(4.17) by forcing its boundary terms to be zero. Two cases are possible: (i) From the given boundary values of $X$ at $t_0$ and $t_f$, if the boundary values of the higher derivatives $\dddot{X}_s, \dddot{X}_s$ can be computed at the two end points, 

$$h = \dddot{h} = \dddot{\dot{h}} = \dddot{\dddot{h}} = 0.$$ 

As a result, $\delta J$ vanishes. In this case, the differential equation in Eq.(4.20) must be solved with boundary values of $X_s, \dddot{X}_s, \dddot{X}_s$ at $t_0$ and $t_f$. (ii) If the values of $n_1, n_2, \ldots, n_p$ are different, then only $\dddot{X}_s, \dddot{X}_s$ can be computed from the
given boundary values of \( X \). In the expression of \( \delta J \), only those elements of \( h \) or its higher derivatives can be crossed off corresponding to which the elements of \( X \) or its higher derivatives are computable. As a result, in the expression of boundary terms of \( \delta J \), some terms remain which must be investigated further to derive potentially extra boundary conditions. On this case further, it can be shown in Appendix B that apart from \( n \) boundary constraints obtained from Eq.(4.12) at two end time, no additional constraint results by forcing the expression of \( \delta J \) to be zero. There is an additional subcase which requires mention: If the maximum and minimum of \((n_1, \ldots, n_p)\) differ by one, all required boundary condition of \( x_{si} \) can be computed by successive use of Eq.(4.12). As a result, all boundary terms in the expression of \( \delta J \) can be crossed off.

### 4.4 Weighted Residual Solution

Different forms of weighted residual methods have been used to solve boundary value problems. A summary of these methods is available [1]. Due to the nature of the optimization problem with fixed end constraints at \( t_0 \) and \( t_f \), only those methods are considered suitable that satisfy the differential equations approximately over the domain but satisfy the boundary conditions exactly such as in Galerkin’s method, method of moments, and collocation method. In this chapter, the method of moments is selected for the solution of the problem since it is computationally simpler than Galerkin’s method. The mode functions are chosen as polynomial functions of time.

#### 4.4.1 Admissible Solution
In order to attain a better understanding of how to choose the admissible solution of $x_{s}$, the elements $x_{s}$ are partitioned into two sets $S_1$ and $S_2$, where $S_1$ contains those elements of $x_{s}$ for which $n_i = N$. In the expression of $J$, the elements of $S_1$ appear up to $N$th derivative while the elements of $S_2$ appear up to $N-1$th derivative. As a result, in the differential equation (4.20), the elements of $S_1$ appear up to $2N$th derivative while the elements of $S_2$ appear up to $2N-2$th derivative. The admissible solution form is taken as:

$$
\begin{bmatrix}
x_{s1} \\
x_{s2} \\
\vdots \\
x_{sp}
\end{bmatrix} =
\begin{bmatrix}
x_{s1a} + \sum_{i=1}^{k} L_{1i} \phi_{1i} \\
x_{s2a} + \sum_{i=1}^{k} L_{21} \phi_{21i} \\
\vdots \\
x_{sp0} + \sum_{i=1}^{k} L_{pi} \phi_{pi}
\end{bmatrix}
$$

(4.21)

where $x_{s0i}$, $i = 1, \ldots, p$ together satisfy the $2n$ boundary conditions of $X_s$ at $t_0$ to $t_f$. $x_{s0i}$ is selected as:

$$x_{s0i} = \sum_{j=0}^{2n-1} F_{yi} t^j$$

(4.22)

where the coefficients $F_{yi}$ are determined from the boundary conditions of $x_{s}$. The $l$th mode function of $\phi_i$ for $t_0 = 0$ is selected as:

$$\phi_{il} = t^{n-1+l} (t - t_f)^n \quad (l = 1, \ldots, k)$$

(4.23)

where $p_i = N$, if $x_{s} \in S_1$, and $N-1$ otherwise. For this choice of mode functions, note that its derivative up to $p_i - 1$ is zero. $L_{il}$ are the mode coefficients to be determined. As a result, $x_{s}$ satisfies the desired boundary conditions regardless of the choice of coefficients $L_{il}$. 
4.4.2 Modal Constants

Eq.(4.21) can be written compactly in the following form:

\[ X_x = X_{x0} + \sum_{i=1}^{k} \Phi_i L_i \]  

(4.24)

where \( X_{x0} = (x_{x10}, x_{x20}, \ldots, x_{xp0})^T \), \( L_i = (L_{i1}, L_{i2}, \ldots, L_{ip})^T \), and \( \Phi_i \) is defined as the following \((p \times p)\) matrix:

\[
\Phi_i = \begin{bmatrix}
\phi_{i1} & 0 & \cdots & 0 \\
0 & \phi_{i2} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \phi_{ip}
\end{bmatrix}
\]  

(4.25)

On substituting Eq.(4.24) in Eq.(4.20), the expression of the error vector, \( \epsilon \), becomes

\[
\epsilon = \sum_{j=0}^{2N} S_j [X_{x0} + \sum_{i=1}^{k} \Phi_i L_i]
\]  

(4.26)

Using the procedure of the weighted residual solution,

\[
\int_{t_0}^{t_f} \epsilon \psi_i(t) dt = 0 \quad (\forall i = 1, \ldots, k)
\]  

(4.27)

where the weighted functions are \( \psi_i = t^{i-1} \). This leads to \( pk \) scalar equations which can be used to solve for the \( pk \) element of the vector \( L_1, \ldots, L_k \). The equations (4.27) can be written in a block matrix form:

\[
\begin{bmatrix}
T_{11} & T_{12} & \cdots & T_{1k} \\
T_{21} & T_{22} & \cdots & T_{2k} \\
\vdots & \vdots & \ddots & \vdots \\
T_{k1} & T_{k2} & \cdots & T_{kk}
\end{bmatrix}
\begin{bmatrix}
L_1 \\
L_2 \\
\vdots \\
L_k
\end{bmatrix} =
\begin{bmatrix}
-R_1 \\
-R_2 \\
\vdots \\
-R_k
\end{bmatrix}
\]  

(4.28)

where \( T_{qi} \) is a \((p \times p)\) matrix subblock, and \( R_q \) is a \((p \times 1)\) vector defined below:
\[
T_q = \int_{t_0}^{2N} S_j \Phi_1(t) \psi_q dt
\]

\[
R_q = \int_{t_0}^{2N} S_j X_{so} \psi_q dt
\]  

(4.29)

The above equation is inverted to solve for the vectors \( L_1, \ldots, L_k \). The expression for \( T_q \) and \( R_q \) are given in Appendix A Eq.(A7) and Eq.(A8).

4.4.3 Recursion with modes

The matrix and vectors of Eq.(4.28) possess certain characteristics which allow the computation of \( L_1, \ldots, L_k \) in a recursive way as the number of modes is increased. Let \( \tau_{k+1} \) be the left-hand side matrix of Eq.(4.28) for \( k+1 \) mode functions:

\[
\tau_{k+1} = \begin{bmatrix}
T_{11} & T_{12} & \cdots & T_{1k} & T_{1k+1} \\
T_{21} & T_{22} & \cdots & T_{2k} & T_{2k+1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
T_{k1} & T_{k2} & \cdots & T_{kk} & T_{kk+1} \\
T_{k+11} & T_{k+12} & \cdots & T_{k+1k} & T_{k+1k+1}
\end{bmatrix}
\]  

(4.30)

It can be shown that \( \tau_{k+1} \) is related to \( \tau_k \) for \( k \) modes in the following way:

\[
\tau_{k+1} = \begin{bmatrix}
\tau_k & \tau_{kk+1} \\
\tau_{k+1k} & \tau_{k+1k+1}
\end{bmatrix}
\]  

(4.31)

where \( \tau_{kk+1} \) is the column block and \( \tau_{k+1k} \) is the row block of \( \tau_{k+1} \). This happens because with increase in number of modes from \( k \) to \( k+1 \), the first \((k \times k)\) block within \( \tau_{k+1} \) is same as \( \tau_k \). Let us denote the solution vector with \( k+1 \) modes as
\[
\ell_{k+1} = \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_k \\ L_{k+1} \end{bmatrix}
\]
(4.32)

This solution vector \( \ell_{k+1} \) can be written from \( \ell_k \) in the following incremental way

\[
\ell_{k+1} = \begin{bmatrix} \ell_k + \delta \ell_k \\ L_{k+1} \end{bmatrix}
\]
(4.33)

Similarly, the right-hand side vector \( \mathcal{R}_{k+1} \) for \( k+1 \) modes is written as

\[
\mathcal{R}_{k+1} = \begin{bmatrix} -R_1 \\ -R_2 \\ \vdots \\ -R_k \\ -R_{k+1} \end{bmatrix}
\]
(4.34)

which can be written from \( \mathcal{R}_k \) in the following incremental way

\[
\mathcal{R}_{k+1} = \begin{bmatrix} \mathcal{R}_k \\ -R_{k+1} \end{bmatrix}
\]
(4.35)

Using matrix properties [6], it can be shown that

\[
\begin{bmatrix} \tau_{k+1} \end{bmatrix}^{-1} = \begin{bmatrix} \tau_k \end{bmatrix}^{-1} + \Delta_2 \Delta_2^{-1} \Delta_3 - \Delta_2 \Delta_1^{-1} \\
-\Delta_1^{-1} \Delta_3 \\
\Delta_1^{-1} \Delta_1^{-1} 
\end{bmatrix}
\]
(4.36)

where

\[
\Delta_1 = T_{k+1,k+1} - \tau_{k+1,k} \begin{bmatrix} \tau_k \end{bmatrix}^{-1} \tau_{k,k+1} \\
\Delta_2 = \begin{bmatrix} \tau_k \end{bmatrix}^{-1} \tau_{k,k+1} \\
\Delta_3 = \tau_{k+1,k} \begin{bmatrix} \tau_k \end{bmatrix}^{-1}
\]
(4.37)

From here, it can be verified that
\[ \delta \ell_k = \Delta_2 \Delta_1^{-1} [\Delta_3 \mathcal{R}_k + R_{k+1}] \]
\[ L_{k+1} = -\Delta_1^{-1} [\Delta_3 \mathcal{R}_k + R_{k+1}] \]  

(4.38)

Eq.(3.33) states that with an increase in the number of modes from \( k \) to \( k+1 \), the solution \( \ell_{k+1} \) can be obtained from \( \ell_k \) in an incremental way. This incremental computation requires inversion of a \((m \times m)\) matrix instead of inverting the full \([m(k+1) \times m(k+1)]\) matrix.

In summary, in order to compute \( \ell_{k+1} \) for \( k+1 \) modes, the following procedure can be adopted: (i) From the solution with \( k \) modes \( \ell_k \), \( \tau_k^{-1}, \mathcal{R}_k \) are available, (ii) From the information of \( k+1 \)th mode, form the matrices \( \tau_{kk+1}, \tau_{k+1k} \) and \( \tau_{k+1k+1} \), (iii) compute \( \delta \ell_k \) and \( L_{k+1} \) from Eq.(4.38) and form \( \ell_{k+1} \).

4.5 Examples

A four degree-of-freedom spring-mass-damper system is used as an example. The system is sketched in figure 11. It consists of eight state variables and four control inputs.

The matrices \( \overline{A} \) and \( \overline{B} \) for this system are as follow:

\[ \overline{A} = \begin{bmatrix} 0 & I_4 \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \quad \overline{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m_1} & 0 & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 & 0 \\ 0 & 0 & \frac{1}{m_3} & 0 \\ 0 & 0 & 0 & \frac{1}{m_4} \end{bmatrix} \]  

(4.39)
where the matrices $M$, $C$, and $K$ are

\[
M = \begin{bmatrix}
    m_1 & 0 & 0 & 0 \\
    0 & m_2 & 0 & 0 \\
    0 & 0 & m_3 & 0 \\
    0 & 0 & 0 & m_4 \\
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
    c_1 + c_2 & -c_2 & 0 & 0 \\
    -c_2 & c_2 + c_3 & -c_3 & 0 \\
    0 & -c_3 & c_3 + c_4 & -c_4 \\
    0 & 0 & -c_4 & c_4 + c_5 \\
\end{bmatrix}
\]

\[
K = \begin{bmatrix}
    k_1 + k_2 & -k_2 & 0 & 0 \\
    -k_2 & k_2 + k_3 & -k_3 & 0 \\
    0 & -k_3 & k_3 + k_4 & -k_4 \\
    0 & 0 & -k_4 & k_4 + k_5 \\
\end{bmatrix}
\] (4.40)

The parameters in MKS units are $m_1 = m_2 = m_3 = m_4 = 1.0$, $c_1 = c_3 = c_5 = 1.0$, $c_2 = c_4 = 2.0$, $k_1 = k_2 = k_3 = k_4 = k_5 = 3.0$. The weighted function $\bar{Q} = 0$ and $R$ is identity of a consistent dimension. The boundary conditions are $\bar{X}(0) = (10, 20, 10, 20, 10, 20, 10, 20)^T$ and $\bar{X}(t_f) = (0, 0, 0, 0, 0, 0, 0, 0)^T$. 
Figure 11: A four degrees-of-freedom spring-mass-damper system.
The solution of four different problems are presented in this chapter: (i) single input $u_1$, (ii) two inputs $u_1$ and $u_3$, (iii) three inputs $u_1$, $u_3$, and $u_4$ with the $T$ matrix consisting of $n_1 = 3, n_2 = 3, n_3 = 2$, (iv) four inputs $u_1, u_2, u_3,$ and $u_4$. In the first problem, the differential equation becomes 16-th order in the variables $x_8$ and the boundary conditions on $x_8$ are known up to 7th derivative at the two end time. In the second problem, the differential equation becomes 8th order in the variables $x_4$ and $x_8$ and the boundary conditions on $x_4$ and $x_8$ are known up to 3rd derivative at the two end time. In the third problem, the differential equation becomes 6th order in $x_3$ and $x_6$ and 4th order in $x_8$. The boundary conditions on $x_3$ and $x_6$ are known up to second derivative and $x_8$ up to 1st derivative at the two end time. In the fourth problem, the differential becomes 4th order in $x_2, x_4, x_6$ and $x_8$. The boundary conditions on $x_2, x_4, x_6$ and $x_8$ are known up to 1st derivative at the two end time. The results of the optimization are plotted respectively in Figure 12, 13, 14, 15. Figure 16 shows a representative plot of the variation of $|\delta \ell_k|$ with iteration steps for problem (iii).
Figure 12: The optimal trajectories for a single input $u_1$ and final time $t_f = 1$ second. The solutions from zero mode, one mode, and matrix exponential method overlap within the accuracy of drawing.

Figure 13: The optimal trajectories for two inputs $u_1$ and $u_3$ and final time $t_f = 1$ second. The solutions from zero mode, one mode, and matrix exponential method overlap within the accuracy of drawing.
Figure 14: The optimal trajectories for three inputs $u_1, u_3, u_4$ with $n_1 = 3$, $n_2 = 3$, $n_3 = 2$ for the final time $t_f = 7$ seconds. The solution are presented for (....) are a zero mode, (_____ ) are five modes, and (_____ ) are ten modes. The ten modes solution is very close to the solution obtained by matrix exponential method.

Figure 15: The optimal trajectories for four inputs $u_1, u_2, u_3$ and $u_4$ and final time $t_f = 1$ second. The solutions from zero mode, one mode, and matrix exponential method overlap within the accuracy of drawing.
Figure 16: A representative plot of the variation of the norm $\delta \ell_k$ with iteration steps $k$ for problem(iii).
4.6 Summary

A new procedure for optimization of linear dynamic systems has been proposed that does not need Lagrange multipliers. In this new method, a linear transformation is used to express the state vector and control inputs in terms of higher derivatives of a subset of state variables. On substituting their expressions in the cost function, the optimization problem reduces to an unconstrained optimization problem where the functional contains higher derivatives. A variational statement of this new functional is written and the necessary conditions for optimality are obtained, which are a higher-order differential equation and the necessary boundary conditions. This differential equation is solved using classical weighted residual methods. It was demonstrated in this chapter that using this new procedure, significant matrix computations are carried out over matrices of dimension \((m \times m)\), as opposed to \((n \times n)\) in conventional procedures. As \(m\) becomes smaller compared to \(n\), the number of computations significantly decreases, giving this procedure a big edge over the conventional computational procedures for determining optimal solutions. Currently, this solution procedure is being extended successfully to solve optimal control problems for time-varying linear systems as well as for nonlinear dynamic equations of robots and manipulators.
Chapter 5

Conclusion

A new method was presented that eliminates the use of Lagrange multipliers for linear time-invariant systems by transformations where the cost functional is expressed as higher derivatives of a subset of the state variables. A variational statement of this cost functional results in the necessary conditions which are higher-order differential equations in fewer variables. These higher-order differential equations are solved using classical weighted residual techniques. This new procedure, solving higher-order optimality equations with weighted residual techniques, requires substantially less computation compared to conventional procedures; therefore, has potential benefits for real time control. The author believes that this new procedure can be extend to time-varying linear systems as well as nonlinear dynamic equations of multi-degree-of-freedom systems.
References


Appendix A

Equation Formulation

\[ T_{pl} = \frac{48(p + l - 1)(l - 1) - 24(p + l - 1)(p + l) - 24(l - 1)(l - 2)}{(p + l - 1)(p + l)(p + l + 1)(p + l + 2)(p + l + 3)} S_4 \]

\[ + \frac{36(p + l)l - 12(p + l)(p + l + 1) - 24l(l - 1)}{(p + l)(p + l + 1)(p + l + 2)(p + l + 3)(p + l + 4)} S_3 \]

\[ + \frac{24(p + l + 1)(l + 1) - 4(p + l + 1)(p + l + 2) - 24(l + 1)l}{(p + l + 1)(p + l + 2)(p + l + 3)(p + l + 4)(p + l + 5)} S_2 \]

\[ + \frac{12(p + l + 2) - 24(1 + l)}{(p + l + 2)(p + l + 3)(p + l + 4)(p + l + 5)(p + l + 6)} S_1 \]

\[ + \frac{-24(-t_f)_{p+7}}{(p + l + 3)(p + l + 4)(p + l + 5)(p + l + 6)(p + l + 7)} S_0 \]

(A1)

\[ R_p = S_3 \frac{-12(-t_f)_{p+4} \Phi_{03}}{(p + 2)(p + 3)(p + 4)} \]

\[ + S_2 \left[ \frac{-4(-t_f)_{p+4} \Phi_{02}}{(p + 2)(p + 3)(p + 4)} + \frac{36(-t_f)_{p+5} \Phi_{03}}{(p + 2)(p + 3)(p + 4)(p + 5)} \right] \]

\[ + S_1 \left[ \frac{-2(-t_f)_{p+4} q_0}{(p + 2)(p + 3)(p + 4)} + \frac{12(-t_f)_{p+5} \Phi_{02}}{(p + 2)(p + 3)(p + 4)(p + 5)} \right] \]

\[ - \frac{72(-t_f)_{p+6} \Phi_{03}}{(p + 2)(p + 3)(p + 4)(p + 5)(p + 6)} \]
\[
S_0 \left[ \frac{-2(-t_f)^{p+4} q_0}{(p+2)(p+3)(p+4)} + \frac{6(-t_f)^{p+5} q_0}{(p+2)(p+3)(p+4)(p+5)} \right] \\
+ \frac{-24(-t_f)^{p+6} \Phi_{\theta_2}}{(p+2)(p+3)(p+4)(p+5)(p+6)} + \frac{120(-t_f)^{p+7} \Phi_{\theta_3}}{(p+2)(p+3)(p+4)(p+5)(p+6)(p+7)}
\] (A2)

where

\[
\Phi_{\theta_2} = \frac{3}{t_f} (q_f - q_0) - \frac{2}{t_f} q_0 - \frac{1}{t_f} q_f
\]
\[
\Phi_{\theta_3} = -\frac{2}{t_f^3} (q_f - q_0) + \frac{1}{t_f^2} (q_f + q_0)
\] (A3)

The expressions for \( \Lambda, U, \) and \( L \) are:

\[
\Lambda = \begin{bmatrix}
0! & 0 & \cdots & 0 & 0 \\
1! & \cdots \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & p-2! & 0 \\
0 & 0 & \cdots & 0 & p-1!
\end{bmatrix}
\]

\[
U = \begin{bmatrix}
0! & t_f & \cdots & t_f^{p-2} & t_f^{p-1} \\
0 & 1! & \cdots & p-2! & t_f^{p-3} & t_f^{p-2} & t_f^{p-1} \\
\vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & p-2! & \frac{p-1!}{1!} t_f \\
0 & 0 & \cdots & 0 & \frac{p-1!}{1!} t_f & \frac{p-1!}{p-1!} t_f
\end{bmatrix}
\]
For mode functions listed in Eq.(3.24)

\[ T_{ql} = \sum_{j=0}^{2p} S_j \left( \sum_{i=0}^{p} (-1)^i C_i^k t_f^i \frac{2p-i+l-1!}{2p-i+l-1-j!} \frac{t^{2p-i+l-1-j+q}}{l_0} \right) \]  

where a symbol \( C_i^k = \frac{k!}{i!(k-i)!} \) and \( j \leq 2p - i + l - 1 \) under the summation to avoid differentiating powers of \( t \), zero or below.

Similarly, the expression for \( R_q \) is:

\[ R_{q1} = \sum_{j=0}^{2p} S_j \left( \sum_{i=0}^{p} F_i \frac{l!}{l-j!} \frac{t^{l-j+q}}{l_0} \right) \quad (j \leq l) \]

\[ R_{q2} = \sum_{j=0}^{2p} S_j \left( \sum_{i=0}^{p} F_{p+i} \sum_{i=0}^{l} (-1)^i C_i^k t_f^i \frac{p+l-i+1!}{p+l-i-j!} \frac{t^{p+l-i-j+q}}{l_0} \right) \]

\[ (j \leq p+l-i) \]

\[ R_q = R_{q1} + R_{q2} \]  

The expression for \( T_{ql} \) in Eq.(4.29) is

\[ T_{ql} = \int \sum_{j=0}^{2N} S_j \Phi_1(t) \psi_p dt = \sum_{j=0}^{2N} S_j \int \Phi_1(t) \psi_p dt \]
The \( r \)th diagonal terms in the above integral are

\[
\sum_{i=0}^{p_r} (-1)^i C_i^r t_i^l \frac{2p_r-i+l-1!}{2p_r-i+l-1-j!} 2p_r-i+l-1-j+q \bigg|_{t_0}^{t_f}
\]

The symbol \( C_i^k = \frac{k!}{i!(k-i)!} \) and \( j \leq 2p_r-i+l-1 \) under the summation to avoid differentiating powers of \( t \), zero or below.

The expression for in Eq.(4.29) is

\[
R_q = \int \sum_{t_0}^{t_f} S_j X_{s_0} \psi \, dt = \sum_{j=0}^{2N} S_j X_{s_0} \psi \, dt
\]

The \( r \)th element of the above integral is

\[
\sum_{i=0}^{2n-r-1} F_i \frac{l!}{l-j!l-j+q} \bigg|_{t_0}^{t_f}
\]
Appendix B

Boundary Condition Terms

In this appendix, we show how all boundary condition terms appearing in Eq.(4.17) are identically zero. First, we rewrite the boundary condition terms appearing in Eq.(4.17) in the following matrix form:

\[
\begin{bmatrix}
\frac{\partial F}{\partial X_s} & \frac{d}{dt} \frac{\partial F}{\partial X_s} & \cdots & \frac{d^{N-1}}{dt^{N-1}} \frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_{s1}} & \frac{d}{dt} \frac{\partial F}{\partial X_{s1}} & \cdots & \frac{d^{N-1}}{dt^{N-1}} \frac{\partial F}{\partial X_{s1}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial F}{\partial X_{sN}} & \frac{d}{dt} \frac{\partial F}{\partial X_{sN}} & \cdots & \frac{d^{N-1}}{dt^{N-1}} \frac{\partial F}{\partial X_{sN}} \\
\end{bmatrix}
\begin{bmatrix}
T^T \\
h^T \\
st \\
\vdots \\
h^T \\
\end{bmatrix}
= 0 \quad \text{(B1)}
\]

The above condition must be satisfied both at \( t_0 \) and \( t_f \). The column vector in Eq.(B1) can be rewritten as
On substituting this notation and defining \( \hat{H}^T = \begin{bmatrix} h^T & h^T & \ldots & h^T & \ldots & h^T \end{bmatrix} \), Eq.(B1) can be rewritten as

\[
\begin{bmatrix}
\frac{\partial F}{\partial \dot{X}_s} \\
\frac{\partial F}{\partial \ddot{X}_s} \\
\frac{\partial F}{\partial \dddot{X}_s} \\
\vdots \\
\frac{\partial F}{\partial X_n}
\end{bmatrix}
- \hat{H}^T \frac{d}{dt} \begin{bmatrix}
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\vdots \\
\frac{\partial F}{\partial X_s}
\end{bmatrix}
+ \hat{H}^T \frac{d^2}{dt^2} \begin{bmatrix}
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\vdots \\
\frac{\partial F}{\partial X_s}
\end{bmatrix}
+ \ldots + (-1)^{N-1} \hat{H}^T \frac{d^{N-1}}{dt^{N-1}} \begin{bmatrix}
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\vdots \\
\frac{\partial F}{\partial X_s}
\end{bmatrix} = 0 \quad (B2)
\]

From Eq.(4.19)

\[
\begin{bmatrix}
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\frac{\partial F}{\partial X_s} \\
\vdots \\
\frac{\partial F}{\partial X_s}
\end{bmatrix}
= C^T QC \begin{bmatrix}
X_s \\
\dot{X}_s \\
\ddot{X}_s \\
\dddot{X}_s \\
\ldots \\
X_n
\end{bmatrix}
\]

and on defining \( W = QC \begin{bmatrix}
X_s \\
\dot{X}_s \\
\ddot{X}_s \\
\dddot{X}_s \\
\ldots \\
X_n
\end{bmatrix} \), Eq.(B2) can be written as follows:

\[
\hat{H}^T C_1^T W - \hat{H}^T C_2^T \ddot{W} + \hat{H}^T C_3^T \dddot{W} + \ldots + (-1)^{N-1} \hat{H}^T C_n^T \frac{d^{N-1}}{dt^{N-1}} \dddot{W} = 0 \quad (B3)
\]
where \( C_1^T \) is the matrix of the same dimension as \( C^T \) starting from its \((m+1)\)th row up to \( mN \)th row with adding \( m \) rows equal to zeros. Similarly \( C_i^T \) is a matrix of the same dimension as \( C^T \) starting from its \((mi+1)\)th row up to \( mN \)th row with adding \( mi \) rows equal to zeros.

On defining \( H_i = \left( u_i, h_{n_1+n_2+\ldots+n_{i-1}+1}, h_{n_1+n_2+\ldots+n_{i-1}+2}, \ldots, h_{n_1+n_2+\ldots+n_{i-1}+1}, h_{n_1+n_2+\ldots+n_1} \right)^T \), from Eq.(4.11)

\[
\begin{bmatrix}
H_1 \\
H_2 \\
\vdots \\
H_p
\end{bmatrix} =
\begin{bmatrix}
h \\
h \\
\vdots \\
h
\end{bmatrix}
\]

(B4)

which all \( h_n \) in the vector \( H_i \) always vanish because of given boundary conditions and also Eq.(B4) can be rewritten as

\[
\begin{bmatrix}
u_1 \\
0 \\
\vdots \\
0 \\
u_2 \\
0 \\
\vdots \\
0 \\
u_m \\
0 \\
0
\end{bmatrix} = C\hat{H}
\]
The above equation show that $C_1$ is the matrix of the same dimension as $C$ using all rows of $C$ which multiply with $\hat{H}$ equal to zero with adding $m$ rows equal to zeros. Similarly $C_i$ is a matrix of the same dimension as $C$ using some rows of $C$ which multiply with $\hat{H}$ equal to zero with adding $m_i$ rows equal to zeros. Therefore, from Eq.(B3), it follows that

$$\hat{H}^T C_1^T = 0, \hat{H}^T C_2^T = 0, \hat{H}^T C_3^T = 0, \ldots, \hat{H}^T C_N^T = 0$$  \hspace{1cm} (B5)$$

In conclusion, from Eq.(B3), we can not find any extra boundary condition, or in other words, all boundary condition terms from Eq.(4.17) are always vanished.
Appendix C
Matlab Program

```
<USER1.M>

%DEFINE tf, X(0), Xdot(0) (BOUNDARY CONDITION)
X0=[10;20;10;20]; %***
Xdot0=[10;20;10;20]; %***
Xf=[0;0;0;0]; %***
Xdotf=[0;0;0;0]; %***

%**************************************************************
cal
%**************************************************************

CAL.M

Y02=3*(Xf-XO)/(tf.^2)-2*XdotO/(tf)-Xdotf/(tf);
Y03=- 2*(Xf-XO)/tf+(Xdotf+XdotO)/tf;

Y03=-2*(Xf-XO)/(tf.^3)+Xdotf+XdotO)/tf; 

Y03=2*(Xf-XO)/(tf.^3) + Xdotf+XdotO)/tf;

%FIND THE RIGHT HAND SIDE MATRIX Rp

for p=1:n,
    R1=12*S3*Y03*((-tf).^p)/(p+4)/(p+5); 
    R2=82*Y02*((-tf).^p)/(p+2)/(p+3)/(p+4); 
    R3=S2*36*Y03*((-tf).^p)/(p+5)/(p+6); 
    R4=81*12*Y02*((-tf).^p)/(p+2)/(p+3)/(p+4); 
    R5=S1*12*Y02*((-tf).^p)/(p+2)/(p+3)/(p+4); 
    R6=S1*(-72)*Y03*((-tf).^p)/(p+6)/(p+7)/(p+8)/(p+9); 
    R7=S0*(-2)*X0*(p+5)/(p+6); 
    R8=S0*6*Xdot0*((-tf).^p)/(p+5)/(p+6); 
    R9=S0*(-24)*Y02*(p+5)/(p+6); 
    %......
```
R10=S0*120*Y03*(-
tf).*((p+7))/((p+2)*(p+3)*(p+4)*(p+5)*(p+6)*(p
+7));
R=(-
1)*((R1+R2+R3+R4+R5+R6+R7+R8+R9+R10);%**
matname=['y',num2str(p)];%**
eval(['matname,=',mat2str(R),',','])
end%for loop

%FIND THE MATRIX Tp1
for i = 1:n,
    for j = 1:n,
        T1=(i+j)*((i+j+1)*(i+j+2)*(i+j+3);
        a=(48*(i+j-1));%**
        b=(24*(i+j-1));%**
        c=(24*(i+j-1));%**
        T2=(a-b-c)*(((i+j)+3));%**
    T3=((i+j+1)*(i+j+2)*(i+j+3);%**
        d=(36*(i+j));%**
        e=(12*(i+j));%**
        f=(24*(i+j));%**
        T4=((d-e-f)*((i+j+4));%**
    T5=((i+j+1)*(i+j+2)*(i+j+3)*1/3;%**
        g=(24*(i+j+1));%**
        h=(24*(i+j+1));%**
        T6=((g-h-k)*((i+j+5));%**
    T7=((i+j+2)+(i+j+3)+1/7;9
        T8=((12*(i+j)+2)*((i-1)*24);%**
    T9=((i+j+3)+(i+j+4)+(i+j+5)+(i+j+6)+(i+j+7);%**
    T10=-24*(((i+j+7));%**
    T=T2+T4+T6+T8+T10;

    matname=['T',num2str(i),',',',',']
eval(['matname,=',mat2str(T),',','])
    end
end

%******************************************************************************
%******************************************************************************
n=2;
t12=[T1J2];%**
t21=[T2J1];%**
t22=T2J2;%**
d0=t22-t21*T1*t12; %*T?
d1=inv(d0);%**
d2=T1*t12; %*T?
d3=t21*T1; %*T?
T2=[T1+2*d1*d2 -d2*d1; %*T?
    -d1*d3 d1];%**
y=[y1];%**add more
dL=d2*d1*(d3*y-y2);%yk+1
dL1=dl; %yl?
d2=dl*(d3*y-y2);%yk+1
L2=[L1+dl; %yl?
    L2];%**
L=L2;
norm(L2);%**
if norm(L2)<=err,
    user2
end

%******************************************************************************
%******************************************************************************
n=3;
t12=[T1J3;T2J3];%**
t21=[T3J1 T3J2];%**
t22=T3J3; %**
d0=t22-t21*T2*t12; %*T?
d1=inv(d0);%**
d2=T2*t12; %*T?
d3=t21*T2; %*T?
T3=[T2+d2*d1*d3 -d2*d1; %*T?
    -d1*d3 d1];%**
y=[y1; y2];%**add more
dL=d2*d1*(d3*y-y3);%yk+1
dL2=dl; %yl?
d2=dl*(d3*y-y3);%yk+1
L3=[L1+dl; %yl?
    L2];%**
L=L3;
norm(L2);%**
if norm(L2)<=err,
    user2
end

%******************************************************************************
%******************************************************************************
n=4;
t12=[T1J4;T2J4;T3J4];%**
t21=[T4J1 T4J2 T4J3];%**
t22=T4J4; %**
d0=t22-t21*T3*t12; %*T?
d1=inv(d0);%**
d2=T3*t12;  
T4=[T3+d2*d1*d3 -d2*d1;  
-yk+1  
T3];  
L4=[L3+dL;  
L2];  
L=L4;  
norm(L2);  
if norm(L2)<=err, user2  
end  

n=5;  
t12=[T1J5;T2J5;T3J5];%**  
t21=[T5J1 T5J2 T5J3 T5J4];%**  
t22=T5J5;  
d0=t22-t21*T5*t12;  
d1=inv(d0);  
d2=T4*t12;  
d3=t21*T4;  
T5=[T4+d2*d1*d3-d2*d1;  
-yk+1  
T4];  
L5=[L4+dL;  
L2];  
L=L5;  
norm(L2);  
if norm(L2)<=err, user2  
end  

n=6;  
t12=[T1J6;T2J6;T3J6;T4J6;T5J6];%**  
t21=[T6J1 T6J2 T6J3 T6J4 T6J5];%**  
t22=T6J6;  
d0=t22-t21*T6*t12;  
d1=inv(d0);  
d2=T5*t12;  
d3=t21*T5;  
T6=[T5+d2*d1*d3-d2*d1;  
-yk+1  
T5];  
L6=[L5+dL;  
L2];  
L=L6;  
norm(L2);  
if norm(L2)<=err, user2  
end  

n=7;  
t12=[T1J7;T2J7;T3J7;T4J7;T5J7;T6J7];%**  
t21=[T7J1 T7J2 T7J3 T7J4 T7J5 T7J6];%**  
t22=T7J7;  
d0=t22-t21*T6*t12;  
d1=inv(d0);  
d2=T6*t12;  
d3=t21*T6;  
T7=[T6+d2*d1*d3-d2*d1;  
-yk+1  
T6];  
L7=[L6+dL;  
L2];  
L=L7;  
norm(L2);  
if norm(L2)<=err, user2  
end  

n=8;  
t12=[T1J8;T2J8;T3J8;T4J8;T5J8;T6J8;T7J8];%**  
t21=[T8J1 T8J2 T8J3 T8J4 T8J5 T8J6 T8J7];%**  
t22=T8J8;  
d0=t22-t21*T7*t12;  
d1=inv(d0);  
d2=T7*t12;  
d3=t21*T7;  
T8=[T7+d2*d1*d3-d2*d1;  
-yk+1  
T7];  
L8=[L7+dL;  
L2];  
L=L8;  
norm(L2);  
if norm(L2)<=err, user2
n=9;
t12=[T1J9;T2J9;T3J9;T4J9;T5J9;T6J9;T7J9;T8J9];
t21=[T9J1 T9J2 T9J3 T9J4 T9J5 T9J6 T9J7 T9J8];
t22=T9J9;
d0=t22-t21*T8*t12; 
d1=inv(d0);
d2=T8*t12; 
d3=t21*T8; 
T9=[T8+d2*d1+d3 -d2*d1]; 

n=10;
t12=[T1J10;T2J10;T3J10;T4J10;T5J10;T6J10;T7J10;T8J10;T9J10];
t21=[T10J1 T10J2 T10J3 T10J4 T10J5 T10J6 T10J7 T10J8 T10J9];
t22=T10J10; 
d0=t22-t21*T9*t12; 
d1=inv(d0);
d2=T9*t12; 
d3=t21*T9; 
T10=[T9+d2*d1+d3 -d2*d1]; 

n=11;
t12=[T1J11;T2J11;T3J11;T4J11;T5J11;T6J11;T7J11;T8J11;T9J11;T10J11];
t21=[T11J1 T11J2 T11J3 T11J4 T11J5 T11J6 T11J7 T11J8 T11J9 T11J10];
t22=T11J11; 
d0=t22-t21*T10*t12; 
d1=inv(d0);
d2=T10*t12; 
d3=t21*T10; 
T11=[T10+d2*d1+d3 -d2*d1]; 

n=12;
t12=[T1J12;T2J12;T3J12;T4J12;T5J12;T6J12;T7J12;T8J12;T9J12;T10J12;T11J12];
t21=[T12J1 T12J2 T12J3 T12J4 T12J5 T12J6 T12J7 T12J8 T12J9 T12J10 T12J11];
t22=T12J12; 
d0=t22-t21*T11*t12; 
d1=inv(d0);
d2=T11*t12; 
d3=t21*T11; 
T12=[T11+d2*d1+d3 -d2*d1]; 

n=13;

t12=[T11J13; T21J13; T31J13; T41J13; T51J13; T61J13; T71J13; T81J13; T91J13; T101J13; T111J13; T121J13]; %
  \* 
  t21=[T13J1; T13J2; T13J3; T13J4; T13J5; T13J6; T13J7; T13J8; T13J9; T13J10; T13J11; T13J12]; %
  \* 
  t22=T13J13; %
  d0=(d2+d1)*T12*t12; %*T?
  d1=inv(d0);
  d2=T12*t12; %*T?
  d3=t21*T12; %*T?
  T13=[T12+d2*d1*d3 -d2*d1]; %*T??
  -d1*d3 d1];
  y=[y1; y2; y3; y4; y5; y6; y7; y8; y9; y10; y11; y12]; %
  *add more
  dL=d2*d1*(d3*y-y13); %yk+1
  dL12=dl;
  l2=(-d1*(d3*y-y13); %yk+1
  L13=[L12+dL; \%L??
  l2];
  L=L13;
  norm(l2);
  if norm(l2)<=err, user2
  end
  %***********************************************************************
  %***********************************************************************
  \* 
  n=14; \%n=?
  t12=[T14J14; T21J14; T31J14; T41J14; T51J14; T61J14; T71J14; T81J14; T91J14; T101J14; T111J14; T121J14]; %
  \* 
  t21=[T14J1; T14J2; T14J3; T14J4; T14J5; T14J6; T14J7; T14J8; T14J9; T14J10; T14J11; T14J12; T14J13]; %
  \* 
  t22=T14J14; %
  d0=(d2+d1)*T13*t12; %*T?
  d1=inv(d0);
  d2=T13*t12; %*T?
  d3=t21*T13; %*T?
  T14=[T13+d2*d1*d3 -d2*d1]; %*T??
  -d1*d3 d1];
  y=[y1; y2; y3; y4; y5; y6; y7; y8; y9; y10; y11; y12; y13]; %
  *add more
  dL=d2*d1*(d3*y-y14); %yk+1
  dL13=dl;
  l2=(-d1*(d3*y-y14); %yk+1
  L14=[L13+dL; \%L??
  l2];
  L=L14;
  norm(l2);
  if norm(l2)<=err, user2
  end
  %***********************************************************************
  %***********************************************************************
  \* 
  n=15; \%n=?
  t12=[T15J15; T21J15; T31J15; T41J15; T51J15; T61J15; T71J15; T81J15; T91J15; T101J15; T111J15; T121J15; T141J15]; %
  \* 
  t21=[T15J1; T15J2; T15J3; T15J4; T15J5; T15J6; T15J7; T15J8; T15J9; T15J10; T15J11; T15J12; T15J13; T15J14]; %
  \* 
  t22=T15J15; %
  d0=(d2+d1)*T14*t12; %*T?
  d1=inv(d0);
  d2=T14*t12; %*T?
  d3=t21*T14; %*T?
  T15=[T14+d2*d1*d3 -d2*d1]; %*T??
  -d1*d3 d1];
  y=[y1; y2; y3; y4; y5; y6; y7; y8; y9; y10; y11; y12; y13; y14]; %
  *add more
  dL=d2*d1*(d3*y-y15); %yk+1
  dL14=dl;
  l2=(-d1*(d3*y-y15); %yk+1
  L15=[L14+dL; \%L??
  l2];
  L=L15;
  norm(l2);
  if norm(l2)<=err, user2
  end
  %***********************************************************************
  %***********************************************************************
  \* 
  n=16; \%n=?
  t12=[T16J16; T21J16; T31J16; T41J16; T51J16; T61J16; T71J16; T81J16; T91J16; T101J16; T111J16; T121J16; T141J16; T151J16]; %
  \* 
  t20=[T16J1; T16J2; T16J3; T16J4; T16J5; T16J6; T16J7; T16J8; T16J9; T16J10; T16J11; T16J12; T16J13; T16J14; T16J15]; %
  \* 
  t21=[T20];
  t22=T16J16; %
  d0=(d2+d1)*T15*t12; %*T?
  d1=inv(d0);
  d2=T15*t12; %*T?
  d3=t21*T15; %*T?
  T16=[T15+d2*d1*d3 -d2*d1]; %*T??
  -d1*d3 d1];
  y=[y1; y2; y3; y4; y5; y6; y7; y8; y9; y10; y11; y12; y13; y14; y15]; %
  *add more
  dL=d2*d1*(d3*y-y16); %yk+1
  dL15=dl;
  l2=(-d1*(d3*y-y16); %yk+1
  L16=[L15+dL; \%L??
  l2];
  L=L16;
  norm(l2);
if norm(l2) <= err, user2
end
%

if norm(l2) <= err, user2
end
%

if norm(l2) <= err, user2
end
%

if norm(l2) <= err, user2
end
%
d1=inv(d0);
d2=T19*t12;  %*T?
d3=t21*T19;  %*T?
T20=[T19+d2*d1*d3 -d2*d1;  %*T??
    -d1*d3 d1];
y=[y1;y2;y3;y4;y5;y6;y7;y8;y9;y10;y11;y12;y13;
y14;y15;y16;y17;y18;y19]; %**add more
D1=d2*t11*(d3*y-y20);  %yk+1
dL19=dL;
L2=[d1*(d3*y-y20);  %yk+1
    L19+dL;  %L??
    L2;]
L=L20;
norm(L2);
if norm(L2)<=err,
user2
end
%******************************************************************************
******************************************************************************
fprintf('Note!, the norm(L2) still larger than err!!\n')
fprintf('Please do the following:\n')
fprintf('Add more mode n=? at line #6 in CAL.m.')
fprintf('Add more Recursion Technic between line that has ****.\n')
fprintf('Or reduce the err int the CAL.m.')

<USER2.M>

n
t=0:tf/tt:tf;
k=t.^2;
a=(t-tf);
fprintf('Please note!\n')
fprintf('After get the plot, if you press enter you will\n')
fprintf('quit the matlab, and also you can not\n')
fprintf('print the plot.\n')
fprintf('If you want to print the plot press\n')
fprintf('CTRL+PAUSE\n')
fprintf('before print the plot\n')

%******************************************************************************
******************************************************************************
for i=1:P,
x11=X0(i)+Xdot0(i)*t+Y02(i)*k+Y03(i)*(t.^3);
v=0;

for j=1:n,
v1=L(i+(j-1)*P)*k.*(a.^(2+j-1));
v2=v1+v;
v=v2;
end
x(i,:)=x11+v;
end
plot(t,x)
grid,pause
quit
%******************************************************************************
******************************************************************************
The above three Matlab program
is used to solve the optimization of linear dynamic systems without Lagrange multipliers in case n = 2m where users must give the informations in only the userl.m by instruction that shown in the program.

<Sample31.m>

%n=number of state, and m=number of input
%Please enter the value of n,m,(p)
n=8;
m=3;
%Please choose your K matrix that equal to 1 by m.
%Must start with each element has the same number = (n/m).
K=[4 2 2];
%please enter your A(bar)
AB=[-5 2 0 0 -6 3 0 0;
2 -3 1 0 3 -6 3 0;
0 1 -3 2 0 3 -6 3;
0 0 2 -3 0 3 -6;
1 0 0 0 0 0 0;
0 1 0 0 0 0;
0 0 1 0 0 0;
0 0 0 1 0 0 0];

%Now enter your B(bar)
BB=[1 0 0;
0 0 0;
0 1 0;
0 0 1;
0 0 0;
0 0 0;
0 0 0;
0 0 0];

%Give the boundary condition at t0 and tf
XB1=[10;20;10;20;10;20;10;20]; %t0
XB2=[0;0;0;0;0;0;0;0]; %tf

%enter the Q matrix
%Please see instruction #0
Q=[1 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0;
0 0 0 1 0 0 0 0;
0 0 0 0 0 0 0 0;
0 0 0 0 0 1 0 0;
0 0 0 0 0 0 0 0];

%enter final time
tf=1;

%enter error that you want
%If you choose your final time less than or
equal to 1,
%please do not try to make the err to be small so
 mucho.
%e.g. 1e-001
err=1e-006

%How many mode that you want?
%Please understand that the increment of
number of mode is 5.
k0=g1;
end

h0=0;
if m==1,
dd=kkk+1;
else
dd=kkk;
end
for i=1:m;
kkk=K(:,i);
for k=1:kkk;

hh=h0;
gl=k+hh;

AD(k,:)=AA(gI,:);
end
for i=1:n;

gt=l+n*(i-l);
AI=[AD;
e(i,:);
eel];
end
for j=1:Kk+1;

fork=l:dd;
gl=k;
g1l=gl+(j-l);
cc(gl,:)=AI(gII,:);
end
fork=l:m;
g0l=k+m*(j-1);
CC(:,g0l)=cc(:,k);
end
for g=1:dd;

g1l=gt+hh;
C(g1l,:)=CC(g,:);
end
h0=kkk+hh;
clear AD loop
end
P=Kk;
M=C*Q*C;

matname=['M',num2str(i),',',',',num2str(j),']
eval([matname,'=',',',mat2str(W),',','])
end
end

if ater character M is a even number, put -
sign before that.

Please see instruction #2

S0=M1J1;
S1=M1J2-M2J1;
S2=M1J3-M2J2+M3J1;
S3=M1J4-M2J3+M3J2-M4J1;
S4=M1J5-M2J4+M3J3-M4J2+M5J1;
S5=-M2J5+M3J4-M4J3+M5J2;
S6=M3J5-M4J4+M5J3;
S7=-M4J5+M5J4;
S8=M5J5;

Please see instruction #3

a=[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
   0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0;
   0 0 6 0 0 0 0 0 -A(4,6)*2 0 0 0 -A(4,8)*2 0 0;
   1 tf tf.^2 tf.^3 tf.^4 tf.^5 tf.^6 tf.^7 0 0 0 0 0 0 0;
   0 0 0 0 0 0 0 1 tf tf.^2 tf.^3 0 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 1 tf tf.^2 tf.^3;
   0 1 2*tf 3*tf.^2 4*tf.^3 5*tf.^4 6*tf.^5 7*tf.^6
   0 0 0 0 0 0 0 0;
   0 0 0 0 0 0 0 1 2*tf 3*tf.^2 0 0 0 0 0 0;
   0 0 0 0 0 0 0 0 0 0 1 2*tf 3*tf.^2;
   0 0 2 6*tf 12*tf.^2 20*tf.^3 30*tf.^4 42*tf.^5
   0 0 0 0 0 0 0 0;
   0 0 6 24*tf 60*tf.^2 120*tf.^3 210*tf.^4 0 0
   -A(4,6)*2 -A(4,6)*6*tf 0 0 -A(4,8)*2 -A(4,8)*6*tf];

al=inv(a);

The following come from matrix
Xdot=AX+Bu after transformation

XA means the boundary conditions at t0 after
transformation

XA2 means the boundary conditions at tf after
transformation

Please see instruction #4

X40=XA(4);
\[ X_{60} = XA(6); \]
\[ X_{80} = XA(8); \]
\[ X_{41} = XA(3) + A(4,4) * XA(4) + A(4,6) * XA(6) + A(4,8) * XA(8); \]
\[ X_{61} = XA(5) + A(6,4) * XA(4) + A(6,6) * XA(6) + A(6,8) * XA(8); \]
\[ X_{81} = XA(7) + A(8,4) * XA(4) + A(8,6) * XA(6) + A(8,8) * XA(8); \]
\[ X_{42} = XA(2) + A(4,4) * XA(4) + A(4,6) * XA(6) + A(4,8) * XA(8); \]
\[ X_{432} = XA(1) + A(4,4) * XA(4) + A(4,6) * XA(6) + A(4,8) * XA(8); \]

\[ b = [X_{40}; X_{60}; X_{80}; X_{41}; X_{61}; X_{81}; X_{42}; X_{432}; 0; 0; 0; 0; 0; 0; 0; 0]; \]
\[ A_I = A(1) * b; \]
\[ A_{11} = [A(1); A(2); A(3); A(4); A(5); A(6); A(7); A(8)]; \]
\[ A_{12} = [A(9); A(10); A(11); A(12); 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0]; \]
\[ A_{13} = [A(13); A(14); A(15); A(16); 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0]; \]

\[ D = A_p(:, 1) * (b(1:end)) + (((((1-1)-(j-1)+p)) / ((1-1)-(j-1)+p)); \]
\[ F = D + 11; \]
\[ l_1 = F; \]
\[ end \]

\[ \text{for } p = 1 : N; \]
\[ \text{for } j = 1 : 2 * p; \]
\[ \text{if } j = 1, \]
\[ k_1 = 1; \]
\[ l_{j_1} = 1; \]
\[ \text{else} \]
\[ k = 1; \]
\[ f = l; \]
\[ l_{j} = 1; \]
\[ \text{end} \]
\[ \text{end} \]

\[ G = S_0 * r_1 + S_1 * r_2 + S_2 * r_3 + S_3 * r_4 + S_4 * r_5 + S_5 * r_6 + S_6 * r_7 + S_7 * r_8; \]
\[ \text{matname} = \text{''R'', num2str(p);} \]
\[ \text{eval([matname, = , mat2str(G), '; ']);} \]
\[ end \]

\[ \text{for } i = 1 : (1-1)-(j-1), \]
\[ l_{j_1} = i * f; \]
\[ f = l; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]

\[ \text{for } i = 1 : (1-1)-(j-1), \]
\[ l_{j_1} = i * f; \]
\[ f = l; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]

\[ \text{for } p = 1 : N; \]
\[ \text{for } j = 1 : 2 * p; \]
\[ \text{if } j = 1, \]
\[ k_1 = 1; \]
\[ l_{j_1} = 1; \]
\[ \text{else} \]
\[ k = 1; \]
\[ f = l; \]
\[ l_{j} = 1; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]

\[ \text{for } i = 1 : (1-1)-(j-1), \]
\[ l_{j_1} = i * f; \]
\[ f = l; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end} \]

\[ \text{end} \]
for p=1:N, % # of modes
    for i=1:m,
        kkk=K(:,i);
        P=kkk;
    end
    for i=1:(kkk+1),
        f=1;
        for j=1:kkk
            ab=j*f;
            f=ab;
        end
        if i==kkk+1,
            lw=1;
        else
            f=1;
            for j=1:(kkk-(i-1)),
                lw=j*f;
                f=lw;
            end
        end
        c(:,i)=ab/(lw1*lw2);
    end
    for i=(kkk+2):40,
        c(:,i)=0; %**DUMMY
    end
end

for i=1:N, % # of modes
    for j=1:2*P+1,
        G2=0;
        for i=1:2*P-(j-1)+(l-1)+1,
            %%%%%%%%%%%%%%%%%%%%%%%%%
            if j==1,
                ab=1;
                lw=1;
            else
                %%%%%%%%%%%%%%%%%%%%%%%%%
                if 2*P-(i-1)+(l-1)-(j-1)==0,
                    lw=1;
                else
                    f=1;
                    for w=1:2*P-(i-1)+(l-1)-(j-1),
                        lw=w*f;
                        f=lw;
                    end
                end
                end
            end
            f=1;
            for w=1:2*P-(i-1)+(l-1)-(j-1),
                lw=w*f;
                f=lw;
            end
        end
        dev=(2*P-(i-1)+(l-1)-(j-1)+p);
        if j>P+1,
            dev=1;
        else
            G=c(:,i)*((-1).^((i-1)))*(tf.^((i-1)))*(ab/lw)*(tf.^((2*P-(i-1)+(l-1)-(j-1)+p))/dev;
            G1=G+G2;
            G2=G1;
        end
    end
end %j loop 1...2P+1
B(:,1)=G1;
end % for I loop

%$%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%NOTE ml=(2*kl+1)*(N), m2=(2*k2+1)*(N),
m3=(2*k3+1)*(N), and so on
%Be careful we must make dummy for m that
has a dimension less than the largest one
for i=(2*k3+2):2*Kk+1,
for j=1:N,
for i=(2*k2+2):2*Kk+1,
for j=1:N,
    m3(i,j)=0; %* _ * * _ * * _*
end
end

for i=1:2*Kk+1,
    fff=[m1(i,:);h;h;];
    matname=['JA',num2str(p),',',num2str(i-1)];
    eval([matname,'=',mat2str(fff),';']);
end

for i=1:2*Kk+1,
    fff=[h,m2(i,:);h;];
    matname=['JB',num2str(p),',',num2str(i-1)];
    eval([matname,'=',mat2str(fff),';']);
end

for i=1:2*Kk+1,
    fff=[h;h;m3(i,:)];
    matname=['JC',num2str(p),',',num2str(i-1)];
    eval([matname,'=',mat2str(fff),';']);
end

end %p loop 1......n

%***************************************************************************
%***************************************************************************
%***************************************************************************

for e=1:N;
    JA10=[JA1j0(:,e)];
    JA11=[JA1j1(:,e)];
    JA12=[JA1j2(:,e)];
    JA13=[JA1j3(:,e)];
    JA14=[JA1j4(:,e)];%* _ * * _ * *
    JA15=[JA1j5(:,e)];
    JA16=[JA1j6(:,e)];
    JA20=[JB2j0(:,e)];
    JA21=[JB2j1(:,e)];
    JA22=[JB2j2(:,e)];
    JA23=[JB2j3(:,e)];
    JA24=[JB2j4(:,e)];
    JA25=[JB2j5(:,e)];
    JA26=[JB2j6(:,e)];
    JA30=[JC3j0(:,e)];
    JA31=[JC3j1(:,e)];
    JA32=[JC3j2(:,e)];
    JA33=[JC3j3(:,e)];
    JA34=[JC3j4(:,e)];
    JA35=[JC3j5(:,e)];
    JA36=[JC3j6(:,e)];
    t1=S0*JA10+S1*JA11+S2*JA12+S3*JA13+S4*JA14+S5*JA15+S6*JA16;
    t2=S0*JA20+S1*JA21+S2*JA22+S3*JA23+S4*JA24+S5*JA25+S6*JA26;
    t3=S0*JA30+S1*JA31+S2*JA32+S3*JA33+S4*JA34+S5*JA35+S6*JA36;
    tp=[t1 t2 t3];
    matname=['tp01',num2str(e)]; %Change here!
    later add tp011
    eval([matname,'=',mat2str(tp),';']);
end %**FOR loop
%***************************************************************************
%***************************************************************************
eval([matname,'=',mat2str(tp),'']);
end

for e=1:N;
  JA10=[JA3j0(:,e)];
  JA11=[JA3j1(:,e)];
  JA12=[JA3j2(:,e)];
  JA13=[JA3j3(:,e)];
  JA14=[JA3j4(:,e)];
  JA15=[JA3j5(:,e)];
  JA16=[JA3j6(:,e)];
  JA20=[JB3j0(:,e)];
  JA21=[JB3j1(:,e)];
  JA22=[JB3j2(:,e)];
  JA23=[JB3j3(:,e)];
  JA24=[JB3j4(:,e)];
  JA25=[JB3j5(:,e)];
  JA26=[JB3j6(:,e)];
  JA30=[JC3j0(:,e)];
  JA31=[JC3j1(:,e)];
  JA32=[JC3j2(:,e)];
  JA33=[JC3j3(:,e)];
  JA34=[JC3j4(:,e)];
  JA35=[JC3j5(:,e)];
  JA36=[JC3j6(:,e)];
end

t1=S0*JA10+S1*JA11+S2*JA12+S3*JA13+S4*JA14+S5*JA15+S6*JA16;
t2=S0*JA20+S1*JA21+S2*JA22+S3*JA23+S4*JA24+S5*JA25+S6*JA26;
t3=S0*JA30+S1*JA31+S2*JA32+S3*JA33+S4*JA34+S5*JA35+S6*JA36;
tp=[t1 t2 t3];
matname=['tp03',num2str(e)];
end

for e=1:N;
  JA10=[JA4j0(:,e)];
  JA11=[JA4j1(:,e)];
  JA12=[JA4j2(:,e)];
  JA13=[JA4j3(:,e)];
  JA14=[JA4j4(:,e)];
  JA15=[JA4j5(:,e)];
  JA16=[JA4j6(:,e)];
  JA20=[JB4j0(:,e)];
  JA21=[JB4j1(:,e)];
end

t20=S0*JA20+S1*JA21+S2*JA22+S3*JA23+S4*JA24+S5*JA25+S6*JA26;
t30=S0*JA30+S1*JA31+S2*JA32+S3*JA33+S4*JA34+S5*JA35+S6*JA36;

t1=S0*JA10+S1*JA11+S2*JA12+S3*JA13+S4*JA14+S5*JA15+S6*JA16;
t2=S0*JA20+S1*JA21+S2*JA22+S3*JA23+S4*JA24+S5*JA25+S6*JA26;
t3=S0*JA30+S1*JA31+S2*JA32+S3*JA33+S4*JA34+S5*JA35+S6*JA36;
tp=[t1 t2 t3];
matname=['tp05',num2str(e)]; %Change here!
ifn! later add tp011
eval(['matname','=','mat2str(tp)',';']);
end %**FOR loop

%*****************************************************************************
*****************************************************************************
%*****************************************************************************
+++++++nn
NN=N
direct
+++++++nn
NN=1;
Tpl1=[tp011];
T1=inv(Tpl1); %****USE RECURSION LATER
L1=T1*(-R1);
L=L1;

%*****************************************************************************
*****************************************************************************
%*****************************************************************************
NN=2;
t12=tp012;
t21=tp021;
t22=tp022;
d0=t22-t21*T1*t12;
d1=inv(d0);
d2=T1*t12;
d3=t21*T1;
T2=[T1+d2*d1*d3 -d2*d1; -d1*d3 d1];
R=[R1];
dl=d2*d1*(d3*(-R)+R2);
dl1=dl;
l2=d1*(d3*(-R)+R2);
l2=[L1+dl;]
l2];
L=L2;
norm(l2);
if norm(l2)<=err,
tvl3
end
%*****************************************************************************
*****************************************************************************

NN=3;
t12=[tp013;tp023]; %**
t21=[tp031 tp032]; %**
t22=tp033; %**
d0=t22-t21*T2*t12; %*T?
d1=inv(d0);
d2=T2*t12; %*T?
d3=t21*T2; %*T?
T3=[T2+d2*d1*d3 -d2*d1; %*T?
    -d1*d3 d1];
R=[R1;R2]; %add more
dl=d2*d1*(d3*(-R)+R3); %Rk+1
dl2=dl;
l2=d1*(d3*(-R)+R3); %Rk+1
L3=[L2+dl;]
l2];
L=L3;
norm(l2);
if norm(l2)<=err,
tvl3
end
%*****************************************************************************
*****************************************************************************

NN=4;
t12=[tp014;tp024;tp034]; %**
t21=[tp041 tp042 tp043]; %**
t22=tp044; %**
d0=t22-t21*T3*t12; %*T?
d1=inv(d0);
d2=T3*t12; %*T?
d3=t21*T3; %*T?
T4=[T3+d2*d1*d3 -d2*d1; %*T?
    -d1*d3 d1];
R=[R1;R2;R3]; %add more
dl=d2*d1*(d3*(-R)+R4); %Rk+1
dl3=dl;
l2=d1*(d3*(-R)+R4); %Rk+1
L4=[L3+dl;]
%L?
NN=5;
t12=[tp015;tp025;tp035;tp045];
t21=[tp051 tp052 tp053 tp054];
t22=tp055;
d0=22-t21*T4*t12;
d1=inv(d0);
d2=T4*t12;
d3=21*T4;
T5=[T4+d2*d1*d3 -d2*d1; d1*d3 d1];
R=[R1;R2;R3;R4];
dL=d2*d1*(d3*(-R)+R5); %Rk+1
L=L4-dL;
l2=-d1*(d3*(-R)+R5); %Rk+1
L5=[L4+dL; dL];
if norm(l2)<=err,
tv3
end
fprintf('NOTE! norm(l2) is still larger than err,
Please do the following:
')
fprintf('Increase the number of err to be larger.
Or
')
fprintf('Add more modes by 5 modes each time
N=5*? in tv3.m\n')

<nnn1.m>
for e=1:N;
JA10=[JA06j0(:,e)];
JA11=[JA06j1(:,e)];
JA12=[JA06j2(:,e)];
JA13=[JA06j3(:,e)];
JA14=[JA06j4(:,e)];
JA15=[JA06j5(:,e)];
JA16=[JA06j6(:,e)];
JA20=[JB06j0(:,e)];
JA21=[JB06j1(:,e)];
JA22=[JB06j2(:,e)];
JA23=[JB06j3(:,e)];
JA24=[JB06j4(:,e)];
JA25=[JB06j5(:,e)];
JA26=[JB06j6(:,e)];
JA30=[JC06j0(:,e)];
JA31=[JC06j1(:,e)];
JA32=[JC06j2(:,e)];
JA33=[JC06j3(:,e)];
JA34=[JC06j4(:,e)];
JA35=[JC06j5(:,e)];
JA36=[JC06j6(:,e)];

 JA40=[JA07j0(:,e)];
 JA41=[JA07j1(:,e)];
 JA42=[JA07j2(:,e)];

 t1=S0*JA10+S1*JA11+S2*JA12+S3*JA13+S4*JA14+S5*JA15+S6*JA16;
t2=S0*JA20+S1*JA21+S2*JA22+S3*JA23+S4*JA24+S5*JA25+S6*JA26;
t3=S0*JA30+S1*JA31+S2*JA32+S3*JA33+S4*JA34+S5*JA35+S6*JA36;

 for e=1:N;
 JA10=[JA7j0(:,e)];
 JA11=[JA7j1(:,e)];
 JA12=[JA7j2(:,e)];
JA13=[JA7j3(:,e)];
JA14=[JA7j4(:,e)];
JA15=[JA7j5(:,e)];
JA16=[JA7j6(:,e)];

JA20=[JB7j0(:,e)];
JA21=[JB7j1(:,e)];
JA22=[JB7j2(:,e)];
JA23=[JB7j3(:,e)];
JA24=[JB7j4(:,e)];
JA25=[JB7j5(:,e)];
JA26=[JB7j6(:,e)];

JA30=[JC7j0(:,e)];
JA31=[JC7j1(:,e)];
JA32=[JC7j2(:,e)];
JA33=[JC7j3(:,e)];
JA34=[JC7j4(:,e)];
JA35=[JC7j5(:,e)];
JA36=[JC7j6(:,e)];

t1=S0*JA10+S1*JA11+S2*JA12+S3*JA13+S4*JA14+S5*JA15+S6*JA16;
t2=S0*JA20+S1*JA21+S2*JA22+S3*JA23+S4*JA24+S5*JA25+S6*JA26;
t3=S0*JA30+S1*JA31+S2*JA32+S3*JA33+S4*JA34+S5*JA35+S6*JA36;

tp=[t1 t2 t3];
matname=['tp07',num2str(e)]; %Change here!
end %**FOR loop

for e=1:N;
JA10=[JA9j0(:,e)];
JA11=[JA9j1(:,e)];
JA12=[JA9j2(:,e)];
JA13=[JA9j3(:,e)];
JA14=[JA9j4(:,e)];
JA15=[JA9j5(:,e)];
JA16=[JA9j6(:,e)];

JA20=[JB9j0(:,e)];
JA21=[JB9j1(:,e)];
JA22=[JB9j2(:,e)];
JA23=[JB9j3(:,e)];
JA24=[JB9j4(:,e)];
JA25=[JB9j5(:,e)];
JA26=[JB9j6(:,e)];

JA30=[JC9j0(:,e)];
JA31=[JC9j1(:,e)];
JA32=[JC9j2(:,e)];
JA33=[JC9j3(:,e)];
JA34=[JC9j4(:,e)];
JA35=[JC9j5(:,e)];
JA36=[JC9j6(:,e)];

t1=S0*JA10+S1*JA11+S2*JA12+S3*JA13+S4*JA14+S5*JA15+S6*JA16;
t2=S0*JA20+S1*JA21+S2*JA22+S3*JA23+S4*JA24+S5*JA25+S6*JA26;
t3=S0*JA30+S1*JA31+S2*JA32+S3*JA33+S4*JA34+S5*JA35+S6*JA36;

tp=[t1 t2 t3];
end %**FOR loop

for e=1:N;
JA10=[JA9j0(:,e)];
JA11=[JA9j1(:,e)];
JA12=[JA9j2(:,e)];
JA13=[JA9j3(:,e)];
JA14=[JA9j4(:,e)];
JA15=[JA9j5(:,e)];
JA16=[JA9j6(:,e)];

JA20=[JB9j0(:,e)];
JA21=[JB9j1(:,e)];
JA22=[JB9j2(:,e)];
JA23=[JB9j3(:,e)];
JA24=[JB9j4(:,e)];
JA25=[JB9j5(:,e)];
JA26=[JB9j6(:,e)];

JA30=[JC9j0(:,e)];
JA31=[JC9j1(:,e)];
JA32=[JC9j2(:,e)];
JA33=[JC9j3(:,e)];
JA34=[JC9j4(:,e)];
JA35=[JC9j5(:,e)];
JA36=[JC9j6(:,e)];
Same with nnn1.m; however, different only the number after JA, JB, and JC
Same with nl31.m; however, it must be changed by following instruction inside nl31.m.
Same with n131.m; however, it must be changed by following instruction inside n131.m.

Same with n131.m; however, it must be changed by following instruction inside n131.m.

Same with n131.m; however, it must be changed by following instruction inside n131.m.

N=NN
L=(-1)*LIO;
o=2;
if o==1,
fprintf('NOTE! Please do the following after we have the coefficients:
')
fprintf('It is depended on N=? on the screen, mean that we have the coefficients
')
fprintf('upto N mode. Then after the line

In each form of u?=L(?)*(L?SJ?);; 
Please make u? until L(?)*(LNJ?) that N
= # of N on screen and S is # of variable
')
fprintf('Also change the summation of u until 
')
fprintf('After you change everything in tv11.m, at second line
')
fprintf('Then run P2 again. GOOD LUCK!
')
fprintf('Strike any key to quit
')
fprintf('Or press Ctrl+Pause to terminate.
')
fprintf('If you use unix system press Ctrl+c to terminate.
')
quit
else

t=0:tf/y;tf;
d=(t-tf);
if \(2^*P-(i-1)+(l-1)-(j-1)\)\(=0,\)
\(lw=1;\)
else
\(f=1;\)
for \(w=1:2^*P-(i-1)+(l-1)-(j-1)\),
\(lw=w*f;\)
\(f=lw;\)
end \%lower!\=lw
end
end

G=C(:,i)*\((i-l)/(I-l)\)*(tit-1)\*(ab11w) * (tit-2)*P.(i-l)+(1-I)-(j-I);\nG1=G+G2;
G2=G1;
end \%G
matname=['L',num2str(I),num2str(l), 'J',num2str(j-l)];
eval(['matname','=',mat2str(G2),';']);
end \%1
end \%1

ul=Al(1)+Al(2).*t+Al(3).*t.^2+Al(4).*t.^3+Al(5).*t.^4+6*Al(6).*t.^5+7*Al(7).*t.^6+Al(8).*t.^7;
u2=L(1)\(=(L1J0);\)
u3=L(4)\(=(L21J0);\)
u4=L(7)\(=(L31J0);\)
u5=L(10)\(=(L41J0);\)
u6=L(13)\(=(L51J0);\)
u7=L(16)\(=(L61J0);\)
u8=L(19)\(=(L71J0);\)
u9=L(22)\(=(L81J0);\)
u10=L(25)\(=(L91J0);\)
u11=L(28)\(=(L101J0);\)
X4\(=ul+u2+u3+u4+u5+u6+u7+u8+u9+u10+u11 ;%+u12+u13+u14+u15+u16;\)

u1=Al(2)+2*Al(3).*t+3*Al(4).*t.^2+4*Al(5).*t.^3+5*Al(6).*t.^4+6*Al(7).*t.^5+7*Al(8).*t.^6;
u2=L(1)\(=(L11J1);\)
u3=L(4)\(=(L21J1);\)
u4=L(7)\(=(L31J1);\)
u5=L(10)\(=(L41J1);\)
u6=L(13)\(=(L51J1);\)
u7=L(16)\(=(L61J1);\)
u8=L(19)\(=(L71J1);\)
u9=L(22)\(=(L81J1);\)
u10=L(25)\(=(L91J1);\)
u11=L(28)\(=(L101J1);\)
X4\(=ul+u2+u3+u4+u5+u6+u7+u8+u9+u10+u11 ;%+u12+u13+u14+u15+u16;\)

u1=Al(3)+6*Al(4).*t+12*Al(5).*t.^2+20*Al(6).*t.^3+30*Al(7).*t.^4+42*Al(8).*t.^5;
u2=L(1)\(=(L11J2);\)
u3=L(4)\(=(L21J2);\)
u4=L(7)\(=(L31J2);\)
u5=L(10)\(=(L41J2);\)
u6=L(13)\(=(L51J2);\)
u7=L(16)\(=(L61J2);\)
u8=L(19)\(=(L71J2);\)
u9=L(22)\(=(L81J2);\)
u10=L(25)\(=(L91J2);\)
u11=L(28)\(=(L101J2);\)
X4\(=ul+u2+u3+u4+u5+u6+u7+u8+u9+u10+u11 ;%+u12+u13+u14+u15+u16;\)

u1=6*Al(4)+24*Al(5).*t+60*Al(6).*t.^2+120*Al(7).*t.^3+210*Al(8).*t.^4;
u2=L(1)\(=(L11J3);\)
u3=L(4)\(=(L21J3);\)
u4=L(7)\(=(L31J3);\)
u5=L(10)\(=(L41J3);\)
u6=L(13)\(=(L51J3);\)
u7=L(16)\(=(L61J3);\)
u8=L(19)\(=(L71J3);\)
u9=L(22)\(=(L81J3);\)
u10=L(25)\(=(L91J3);\)
u11=L(28)\(=(L101J3);\)
X4\(=ul+u2+u3+u4+u5+u6+u7+u8+u9+u10+u11 ;%+u12+u13+u14+u15+u16;\)

u1=Al(1)+Al(2).*t+Al(3).*t.^2+Al(4).*t.^3+Al(5).*t.^4+6*Al(6).*t.^5+7*Al(7).*t.^6+Al(8).*t.^7;
u2=L(1)\(=(L11J0);\)
u3=L(4)\(=(L21J0);\)
u4=L(7)\(=(L31J0);\)
u5=L(10)\(=(L41J0);\)
u6=L(13)\(=(L51J0);\)
u7=L(16)\(=(L61J0);\)
u8=L(19)\(=(L71J0);\)
u9=L(22)\(=(L81J0);\)
u10=L(25)\(=(L91J0);\)
u11=L(28)\(=(L101J0);\)
X4\(=ul+u2+u3+u4+u5+u6+u7+u8+u9+u10+u11 ;%+u12+u13+u14+u15+u16;\)
The all above program is the Matlab program that is used to solve the optimization of linear dynamic systems.
without Lagrange multipliers in case $n = pm$ and $n = Nm$ by follow the instruction.
Instruction: (shown with example 2 in chapter 2)

#0 How to form matrix $Q$: (Choose $\overline{Q} = 0$)

In the example in chapter 2, we need to optimize with the cost functional

$$u_1^2 + u_3^2$$

We have $X_1$ and $X_2$ in the following form:

$$X_1 = \begin{bmatrix} u_1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \\
X_2 = \begin{bmatrix} u_3 \\ x_3 \\ x_6 \\ x_7 \end{bmatrix}$$

then the cost function become

$$J = \int_t^T \begin{bmatrix} u_1 \\ x_1 \\ x_2 \\ x_3 \\ u_3 \\ x_3 \\ x_6 \\ x_7 \end{bmatrix}^T \begin{bmatrix} u_1 \\ x_1 \\ x_2 \\ x_3 \\ u_3 \\ x_3 \\ x_6 \\ x_7 \end{bmatrix} dt$$

that we can find the matrix $Q$.

#1 How to form matrix $W$

On line above this place, we have matrix $M$ that equal to

$(m(p+1) \times m(p+1))$.

In example $M = (10 \times 10)$ matrix.

The way to write $W$ matrix is depend on number of inputs.

If $m = 2$, $W = (2 \times 2)$. 
If $m = 3$, $W = (3 \times 3)$.

For example $m = 2$;

$$W = \begin{bmatrix} M_{(i+(i-1),j+(j-1))} & M_{(i+(i-1),j+(j-0))} \\ M_{(i+(i-0),j+(j-1))} & M_{(i+(i-0),j+(j-0))} \end{bmatrix}$$

$m = 3$;

$$W = \begin{bmatrix} M_{(i+(2i-2),j+(2j-2))} & M_{(i+(2i-2),j+(2j-1))} & M_{(i+(2i-2),j+(2j-0))} \\ M_{(i+(2i-1),j+(2j-2))} & M_{(i+(2i-1),j+(2j-1))} & M_{(i+(2i-1),j+(2j-0))} \\ M_{(i+(2i-0),j+(2j-2))} & M_{(i+(2i-0),j+(2j-1))} & M_{(i+(2i-0),j+(2j-0))} \end{bmatrix}$$

#2 After #1 matrix $M$ is become:

In example

$$M = \begin{bmatrix} M_{1J_1} & M_{1J_2} & M_{1J_3} & M_{1J_4} & M_{1J_5} \\ M_{2J_1} & M_{2J_2} & M_{2J_3} & M_{2J_4} & M_{2J_5} \\ M_{3J_1} & M_{3J_2} & M_{3J_3} & M_{3J_4} & M_{3J_5} \\ M_{4J_1} & M_{4J_2} & M_{4J_3} & M_{4J_4} & M_{4J_5} \\ M_{5J_1} & M_{5J_2} & M_{5J_3} & M_{5J_4} & M_{5J_5} \end{bmatrix}$$

Make summation like below:

In example

$$S_0 = M_{1J_1}$$
$$S_1 = M_{1J_2} - M_{2J_1}$$
$$S_2 = M_{1J_3} - M_{2J_2} + M_{3J_1}$$
$$S_3 = M_{1J_4} - M_{2J_3} + M_{3J_2} - M_{4J_1}$$
$$S_4 = M_{1J_5} - M_{2J_4} + M_{3J_3} - M_{4J_2} + M_{5J_1}$$
$$S_5 = -M_{2J_5} + M_{3J_4} - M_{4J_3} + M_{5J_2}$$
$$S_6 = M_{3J_5} - M_{4J_4} + M_{5J_3}$$
$$S_7 = -M_{4J_5} + M_{5J_4}$$
$$S_8 = M_{5J_5}$$

#3 How to form matrix $a$ (square matrix)
Each rows come from the same polynomial that up to $2p - 1$.

In example $p = n/m = 4$

Then $X = 1 + t + t^2 + t^3 + t^4 + t^5 + t^6 + t^7$

Therefore;

$$
\begin{bmatrix}
  x(t_o) \\
  x(t_0) \\
  x(t_0) \\
  x(t_0) \\
  x(t_0) \\
  x(t_f) \\
  x(t_f) \\
  x(t_f) \\
  x(t_f) \\
\end{bmatrix}_{(2p \times 2p)}
$$

#4 This is the boundary conditions that we need after transformation at the two end time.

#5 How to form matrix $Ap$

$Ap$ is $(m \times 1)$ matrix.

The element is $A_{Im}$ transpose.

In example, we have $Al_1$ and $Al_2$ then

$$Ap = \begin{bmatrix} A_{l_1}^T; A_{l_2}^T \end{bmatrix}$$

#6 $G = S_0 * r_1 + ... + S_{j-1}r_j$, and also depended on how many $S_j$ that we have.

#7 This place also depends on the number of inputs that we have to make for-loop equal to $m$ loops, and $ff$ is $(m \times 1)$ matrix that must be changed each loop.
For example;

If \( m = 2 \), in loop 1

\[
ff = \begin{bmatrix}
BBB(i,:) \\
h
\end{bmatrix}
\]

in loop 2

\[
ff = \begin{bmatrix}
h \\
BBB(i,:) \\
h
\end{bmatrix}
\]

If \( m = 3 \), in loop 1

\[
ff = \begin{bmatrix}
BBB(i,:) \\
h \\
h
\end{bmatrix}
\]

in loop 2

\[
ff = \begin{bmatrix}
h \\
BBB(i,:) \\
h
\end{bmatrix}
\]

in loop 3

\[
ff = \begin{bmatrix}
h \\
h \\
BBB(i,:) \\
h
\end{bmatrix}
\]

Also at the line number 4 in each loop, we must change string variable \( JA \) in the first loop to be \( JB \) in the second loop and so on.

#8

\[
JA(m)(p) = [\nabla N_j(p)(:,e)];
\]

\[
t(m) = s(p) * JA(m)(p);
\]
$m =$ the number of inputs

$p =$ from 0 to $2p$

$\nabla =$ strings that are changed from #7

$N =$ number of modes

\[ tp = [t_1 \ t_2 \ldots \ldots]_{(1 \times m)} \]