Fall 2019

Extension of a Penalty Method for Numerically Solving Constrained Multibody Dynamic Problems

Troy Newhart
Old Dominion University, tsn5048@gmail.com

EXTENSION OF A PENALTY METHOD FOR NUMERICALLY

SOLVING CONSTRAINED MULTIBODY DYNAMIC PROBLEMS

by

Troy S. Newhart
B.S. May 2015, The Pennsylvania State University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

AEROSPACE ENGINEERING

OLD DOMINION UNIVERSITY
December 2019

Approved by:

Gene Hou (Advisor)

Brett Newman (Member)

Miltos Kotinis (Member)

ABSTRACT

EXTENSION OF A PENALTY METHOD FOR NUMERICALLY
SOLVING CONSTRAINED MULTIBODY DYNAMIC PROBLEMS

Troy Newhart
Old Dominion University, 2019
Director: Dr. Gene Hou

Numerical analysis of constrained static and multibody dynamic systems has become an integral part of engineering analysis with the continued improvements in technology and software availability. Many methods currently exist for numerically solving constrained static and dynamic systems. The applicability of a penalty method for constrained static solutions is observed in many academic texts and papers. The appeal for using a penalty method in statics pertains to its ease of implementation, computational suitability, and accuracy. This thesis extends a static penalty method for use with constrained multibody dynamics to observe if the penalty method's benefits are similar for a dynamics solution.

This thesis discusses formulations that are used in extending the static penalty method for use with constrained multibody dynamics. Example problems are then solved using the static penalty method and compared with a projection method. Example problems are selected to provide a solid foundation for implementing the static penalty method with many other constrained multibody systems. Constraints are purely holonomic for simplification of problem statements.

The goal of this thesis was met, in that the static penalty method is successfully applied to constrained multibody systems with favorable results. In comparing the penalty method with a projection method for each example, computational time and accuracy are comparable.

Coding of the penalty method is found to be no more difficult than that of the projection method. The static penalty method is shown to be useful in solving constrained multibody dynamics. Application of the penalty method was not tested with non-holonomic constraints. Future work is necessary to assess the penalty method's applicability with non-holonomic constraints, as well as increasingly complex multibody dynamic systems.

This thesis is dedicated to my friends and family. It is your love and support that has made me who I am today. To my wife who motivates me to be a better person each and every day. In memory of my mom, your words of encouragement were thought of through the tough times. You are loved and missed.

ACKNOWLEDGMENTS

I would like to thank Dr. Gene Hou for his guidance and patience throughout this research. Dr. Hou's commitment to teaching has been inspirational and motivating toward completing this work. I would also like to thank Dr. Brett Newman and Dr. Miltos Kotinis for their help with this research. All of these gentlemen deserve a thank you for taking the time to review and edit this thesis.

NOMENCLATURE

a	Known static displacement
A	Transformation matrix
α	Baumgarte velocity weighting coefficient
$\boldsymbol{\alpha}'$	Angular accelerations in body-fixed reference frame
b	Body thickness
B	Static body constraint matrix
β	Static body constraints
C	Static penalty factor
$\boldsymbol{C}$	Dynamic constraint vector
C_c	Static attachment constraint vector
$\boldsymbol{C}_{,\boldsymbol{q}}$	Dynamic constraint Jacobian
$\boldsymbol{C}_{,\boldsymbol{q}t}$	Dynamic constraint Jacobian time derivative
$\boldsymbol{C}_{,t}, \boldsymbol{C}_{,tt}$	Dynamic constraint first- and second-time derivatives
dt	Change in time
e_0, e_1, e_2, e_3	Euler parameters
E	Euler E matrix
$\boldsymbol{f_b}$	Body force vector
$\boldsymbol{F}$	Force vector
g	Gravitational constant
G	Euler G matrix
$\boldsymbol{\gamma}$	DAE index constraint formulation

h	Runge-Kutta time step
I	Identity matrix
J'	Mass moment of inertia in body-fixed coordinates
k	Runge-Kutta parameter
$\boldsymbol{K}$	Static stiffness matrix
$\boldsymbol{K_c}$	Constrained static stiffness matrix
L	Body length
$\boldsymbol{\lambda}$	Lagrange multiplier
m	Body mass
$\boldsymbol{M}$	Mass matrix
μ	Dynamic penalty factor
$\boldsymbol{\omega}', \widetilde{\boldsymbol{\omega}}'$	Angular velocity vector, skew matrix in body-fixed reference frame
$\boldsymbol{p}$	Euler parameter vector
$\boldsymbol{P}$	Projection Formulations
Π	Potential Energy
$\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$	Dynamic motion displacement, velocity, and acceleration
Q	Static displacement vector
$\boldsymbol{r}, \dot{\boldsymbol{r}}, \ddot{\boldsymbol{r}}$	Position, velocity, and acceleration in global coordinates
$\boldsymbol{r}'$	Position from body-fixed origin to point of interest
$\boldsymbol{R}$	Position of body-fixed origin from global origin
$\boldsymbol{R}_c', \widetilde{\boldsymbol{R}}_c'$	Position to point from body-fixed centroid vector, skew matrix
t	Time

t_{final}	Final time
t_{init}	Initial time
$\boldsymbol{T_{r'}}$	External torques in the body-fixed reference frame
$\boldsymbol{T_R}$	External forces in global coordinates
$\boldsymbol{\tau}$	Torque
U	Strain energy
ζ	Baumgarte displacement weighting coefficient
x, y, z	Cartesian coordinates
θ, ϕ, ψ	Euler angles

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 BACKGROUND

The application of a penalty method is commonly utilized for the analysis of static mechanisms where penalty factors are applied to constraint parameters of motionless systems. Large penalties are placed on parameters associated with the constraints of a system, and these modified parameters result in calculated values that have been restricted at the boundary conditions as intended. Using a penalty method for static analysis is beneficial due to its simplicity, ease of application, and computational suitability.

Although the use of a penalty method is beneficial for static analysis, its application for constrained multibody dynamics is infrequent. There are cases where a penalty method is utilized with ordinary differential equations (ODE) to solve system kinematics, but formulating constrained multibody dynamic equations of motion (EOM) as an ODE is not always possible or optimal. For constrained EOMs, differential algebraic equations (DAE) are generally formulated. A DAE has characteristics of both algebraic and differential equations that must be solved in tandem to ensure constraints are adhered to. Due to these algebraic constraints, solutions cannot be found by strictly using ODE methods.

There are numerous methods for solving DAEs [2, 7, 8, 10, 12]. To the knowledge of the author, the application of a penalty method to the constraints of a DAE is one that has not been tried against other more popular methods. Although there are various methods, all have their own restrictions and assumptions that must be considered. In the event that this penalty

method does not result in faster or more accurate solutions than current methods, there is always the potential that its application is for a special case that befuddles common methods. It is also possible that this method may be easier to implement within computer code.

The modelling of dynamic systems has become an integral part of industry as computational technology continues to advance along with the development of accurate and affordable software suites. The use of computational modelling for dynamic systems provide users with data and graphics that are expected by consumers in most industries. Accuracy and speed of solution will be expected to progress even though problems being computationally modelled continue to increase in complexity and size. This is especially true in the analysis of constrained multibody dynamics. As the need for computationally stable, accurate, and efficient solutions grow, it is necessary to ensure that adequate research has been conducted towards developing a toolbox composed of various methods that can be utilized to solve increasingly complex problems of the future.

The objective of this thesis is to apply an extension of a penalty method commonly used with static analysis to constrained multibody dynamic systems. The ease with which this method may be applied to constrained multibody dynamic systems will be examined, as coding solutions for such problems can be difficult. The first chapter covers the background and will discuss the literature review of this topic. Equations of motion, ordinary differential equations, and differential-algebraic equation literature is reviewed first. Literature is then presented on solution methods for multibody dynamics that solve ordinary differential equations and differential algebraic equations numerically. Literature on penalty methods used for static and ODE solutions is then presented. Chapter two begins with the derivation of a penalty method

for solving differential-algebraic equations (DAEs), followed by an overview of MATLAB® software used to perform the penalty method analysis on example problems. Example problems are then illustrated to compare the penalty method with a projection method [12]. Example problems include a problem statement, initial conditions, formulation of Equations of Motion (EOM), formulation of DAEs, and results using the penalty method. The example problems depict the use of a penalty method with a simple pendulum, a Scotch mechanism, and a slider-crank mechanism. Chapter three provides a brief discussion of results from the example problems. An overview of the penalty methods limitations and uses for multibody dynamics is then explained. Future work required on this topic concludes the paper.

The penalty method used in this work for dynamic situations is an extension of a penalty method that is commonly utilized for static analysis. The application of the penalty method in static scenarios is useful in that it is easily applied to a variety of static problems, accurately models desired parameters, and is easily programmable for computational modelling. Given these attributes for static problems, it was expected that applying a similar method for dynamic scenarios would have coincidental benefits. Ultimately, the author pursued a method that could be implemented more easily in coding such that modelling constrained multibody dynamic problems requires less work for both the computer and user. It was foreseen that additional research would be required to apply this method to other transient events (i.e. heat transfer, fluid flow) given the scope of this paper. Future work may also be performed to apply the presented method to advancing levels of dynamic complexity, as additional bodies and constraints may be added to model more involved systems than those examined in this work.

1.2 LITERATURE REVIEW

1.2.1 EQUATION OF MOTION FORMULATION

The equations of motion (EOMs) represent the dynamics of a system mathematically. The EOMs are composed of variables that relate physical aspects of a system to each other as a function of time. For a kinematic approach, the common variables used to explain a system's motion with respect to time are the spatial displacements, velocities, and accelerations. The formulation of the kinematic equations can be done for multibody systems that contain relative coordinate systems. In all dynamic systems, a global reference frame is applied to define absolute coordinates. Each body (if multiple bodies exist) can then be assigned its own body-fixed coordinates. Kinematic equations are then applied to relate the body-fixed coordinates to the global coordinates. Ongoing research in determining a method for optimal placement of coordinates describing multibody dynamics is illustrated in [1].

The position of a point in global coordinates can be described by equation (1).

$$\boldsymbol{r} = \boldsymbol{R} + A\boldsymbol{r}' \tag{1}$$

Once the position vectors are formulated, velocity and acceleration can be found by differentiating equation (1) with respect to time. After obtaining the kinematic equations, the kinetic equations can be derived. The kinetic equations formulate a system's EOMs. Using Newton's 2nd Law the EOMs of a single lumped mass are:

$$\boldsymbol{F} = m\ddot{\boldsymbol{r}} \tag{2}$$

$$\boldsymbol{\tau} = J\boldsymbol{\alpha}' \tag{3}$$

For simplicity, individual bodies will be assumed as rigid. The EOMs of an unconstrained free rigid body can then be formulated from [2] as equations (4) and (5) where the body-fixed coordinates are not necessarily at the rigid body's centroid.

$$\boldsymbol{M} = \begin{bmatrix} mI & (-mA\widetilde{\boldsymbol{R}}_{\boldsymbol{c}}')^T \\ (-mA\widetilde{\boldsymbol{R}}_{\boldsymbol{c}}') & J' \end{bmatrix} \tag{4}$$

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{T_R} + \boldsymbol{f_b} - mA\widetilde{\boldsymbol{\omega}}'\widetilde{\boldsymbol{\omega}}'\boldsymbol{R}_{\boldsymbol{c}}' \\ \boldsymbol{T}_{\boldsymbol{r}'} + \widetilde{\boldsymbol{R}}_{\boldsymbol{c}}' A^T \boldsymbol{f_b} - \widetilde{\boldsymbol{\omega}}' J' \boldsymbol{\omega}' \end{bmatrix} \tag{5}$$

It is important to note that the force terms for translation of the system are expressed in the global coordinate frame, whereas the rotational components are in the body-fixed reference frames. If the acceleration terms for translation and rotation are denoted as $\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{\boldsymbol{r}} \\ \boldsymbol{\alpha} \end{bmatrix}$, combining equations (4) and (5) with (2) yields the final EOM for an unconstrained free rigid body:

$$\boldsymbol{F} = \boldsymbol{M}\ddot{\boldsymbol{q}} \tag{6}$$

The above EOMs can be extended to a system of unconstrained free rigid bodies characterized by its translational and rotational motion. As additional unconstrained free bodies are considered, the matrices are combined to depict the EOMs of an entire unconstrained multibody system. The multibody components of equation (6) become

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_1 & 0 & \cdots & 0 \\ 0 & \boldsymbol{M}_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & \boldsymbol{M}_n \end{bmatrix} \tag{7}$$

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{\boldsymbol{q}}_1 \\ \ddot{\boldsymbol{q}}_2 \\ \vdots \\ \ddot{\boldsymbol{q}}_n \end{bmatrix} \tag{8}$$

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{F}_1 \\ \boldsymbol{F}_2 \\ \vdots \\ \boldsymbol{F}_n \end{bmatrix} \tag{9}$$

The EOMs' defining physical parameters of an unconstrained free multibody system can now be formulated. These EOMs are composed of differential variables with respect to time. Formulations containing such differentials are known as Ordinary Differential Equations (ODEs).

1.2.2 ORDINARY DIFFERENTIAL EQUATIONS

Ordinary differential equations are commonly formulated to describe the dynamics of a system. The derivation and application of ODEs cover a wide range of topics not limited to multibody dynamics and are explained in numerous academic texts. Even when the topic range is narrowed to multibody dynamics, many references are available to interested readers. An ODE is classified as a differential equation containing derivatives with respect to a single dependent variable [3, 4]. The dependent variable tends to be time when defining an ODE for physical processes. ODEs can be classified by their "order" where the order is equal to the number associated with the highest derivative.

$$\dot{y} - Cy = 0 \tag{10}$$

$$\ddot{y} + K\dot{y} - Cy = 0 \tag{11}$$

Equations (10) and (11) depict the form of a first order and second order ODE respectively, where y is an arbitrary variable that is a function of time. C and K are arbitrary

constants. In this work, all example problems will be given initial conditions at time equal to zero. Initial value ODEs are commonly defined due to practical formulations that do not have closed-form solutions.

It is always ideal to find a closed-form solution to an ODE that allows for exact analytical calculations to be derived. In practical applications, this is seldom possible and numerical methods are necessary. Numerical methods estimate the parameters of an ODE over a specified time span given a time step. One ODE numerical method of particular interest for this research is the Runge-Kutta Method. The Runge-Kutta method has multiple approaches that may be utilized. Selecting an approach is based on the order of the ODE, desired computational time, and stiffness [5] of the problem. In general, the Runge-Kutta method is an explicit numerical ODE solving algorithm that can be derived for higher-order systems. A solution method of particular interest for this research is the Bogacki-Shampine Runge-Kutta method [15] depicted in equations (12-17).

$$k_1 = f(t_n, y_n) \tag{12}$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \tag{13}$$

$$k_3 = f(t_n + \frac{3h}{4}, y_n + \frac{3h}{4}k_2) \tag{14}$$

$$y_{n+1}^{(1)} = y_n + \frac{2}{9}hk_1 + \frac{1}{3}hk_2 + \frac{4}{9}hk_3 \tag{15}$$

$$k_4 = f(t_n + h, y_{n+1}) \tag{16}$$

$$y_{n+1}^{(2)} = y_n + \frac{7}{24}hk_1 + \frac{1}{4}hk_2 + \frac{1}{3}hk_3 + \frac{1}{8}hk_4 \tag{17}$$

where $y_{n+1}^{(1)}$ is the third-order approximation of the ODE and $y_{n+1}^{(2)}$ is the second-order approximation. The difference between the third-order and second-order is utilized to change step size accordingly. The next k_1 is then set equal to k_4 at the start of the next sequence.

Given a set of initial conditions, Runge-Kutta methods are utilized for integration of a specified ODE. This integration provides robust and adapted solutions to an ODE. The Bogacki-Shampine Runge-Kutta method will be utilized computationally for subsequent example problems in this thesis via the MATLAB® ODE23 three stage, third order Runge-Kutta function. Unfortunately, not all physical processes can be completely described using an ODE. When dealing with constrained multibody dynamics, it is common that resulting formulations are in the form of differential-algebraic equations (DAEs). DAEs cannot be solved using only ODE techniques [6].

1.2.3 DIFFERENTIAL-ALGEBRAIC EQUATIONS AND CONSTRAINT FORMULATION

Differential-Algebraic Equations have properties of both differential equations and algebraic equations, both of which must be solved simultaneously to accurately model the system. DAEs are commonly derived for constrained multibody dynamics due to the differential nature of the EOMs and the algebraic nature of enforcing constraint equations. Although DAEs will be formulated explicitly for constrained multibody dynamics in this paper, DAEs appear in many other disciplines as well.

Constraint equations are imposed to ensure that the dynamics adhere to specified parameters that are otherwise not captured in the EOMs. Constraints can be defined as holonomic or non-holonomic and rheonomous or scleronomous [7]. Holonomic constraints are defined as equality constraints applied to system displacements and will be solely used in this research for simplicity of DAE formulations. Constraints can also explicitly contain time as a variable (rheonomous) or expressed with variables other than time (scleronomous). DAEs are also defined by the number of derivations required to convert a DAE to an ODE. For the duration of this paper, index 3 DAEs will be the primary focus of the research due to their frequency of appearance in constrained multibody dynamics [8].

Once a holonomic constraint has been defined, it must be applied with the EOM of the system to accurately model the dynamics. The DAE is formulated using equations (7), (8), and (9) as [2, 8, 9, 10]:

$$\boldsymbol{M\ddot{q} - F + C_{,q}{}^{T}\lambda = 0} \tag{18}$$

$$\boldsymbol{C(q,t) = 0} \tag{19}$$

where $\boldsymbol{C_{,q}}$ is the Jacobian matrix of the constraint equations, illustrated as equation (20), and $\boldsymbol{\lambda}$ is unknown Lagrange multipliers used to enforce constraints. The Lagrange multipliers in equation (18) are more commonly referred to as the generalized constraint forces. These generalized constraint forces are the link between the forces acting on constraints and the forces acting externally on the bodies. If no constraints are present, we see that the equation reverts back to its unconstrained form depicted in equation (6).

$$\boldsymbol{C}_{,\boldsymbol{q}} \equiv \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{q}} \tag{20}$$

The DAE can then be presented in matrix form as

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{C}_{,\boldsymbol{q}}^{T} \\ \boldsymbol{C}_{,\boldsymbol{q}} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{\gamma} \end{bmatrix} \tag{21}$$

which enforces a constraint on the second order derivative of the constraints, or mathematically $\boldsymbol{C}_{,\boldsymbol{q}}\ddot{\boldsymbol{q}} = \boldsymbol{\gamma}$, where $\boldsymbol{\gamma}$ is

$$\boldsymbol{\gamma} \equiv -\left(\boldsymbol{C}_{,\boldsymbol{q}}\dot{\boldsymbol{q}}\right)_{,\boldsymbol{q}}\dot{\boldsymbol{q}} - 2\boldsymbol{C}_{,\boldsymbol{q}t}\dot{\boldsymbol{q}} - \boldsymbol{C}_{,tt} \tag{22}$$

Since equation (21) only enforces a constraint on $\ddot{\boldsymbol{C}} = \boldsymbol{0}$, the conditions of $\dot{\boldsymbol{C}} = \boldsymbol{0}$ and $\boldsymbol{C} = \boldsymbol{0}$ may not be met. Therefore, the solution to equation (21) must also satisfy the following hidden constraints of equations (23) and (24):

$$\boldsymbol{C} = 0 \tag{23}$$

$$\boldsymbol{C}_{,\boldsymbol{q}}\dot{\boldsymbol{q}} + \boldsymbol{C}_{,t} = 0 \tag{24}$$

Another formulation of an Index-3 DAE that allows for computation without explicitly solving the hidden constraints is presented in [17]. The Baumgarte stabilization method presents additional terms in the force vector that correct constraint errors as the DAE is solved. Equation (25) depicts an Index-3 DAE with the Baumgarte constraint applied, where α and ζ are user applied weighting coefficients.

$$\begin{bmatrix} M & \left(\boldsymbol{C}_{,\boldsymbol{q}}\right)^{T} \\ \boldsymbol{C}_{,\boldsymbol{q}} & 0 \end{bmatrix} \begin{Bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{F} \\ -\boldsymbol{\gamma} - \mathbf{2}\alpha\dot{\boldsymbol{C}} - \zeta^{2}\boldsymbol{C} \end{Bmatrix} \tag{25}$$

In special cases of spatial multibody dynamics, it may be necessary (or more computationally efficient) to employ the use of Euler parameters as opposed to Euler angles when avoiding transformation matrix singularities [8]. The use of Euler parameters is advantageous in that four parameters are utilized to formulate the transformation matrix of rotation, eliminating the potential for singular transformation matrices that can occur when using Euler Angles. The major disadvantage of Euler parameters is the required constraint that must be adhered to when applying these parameters to the EOMs. Euler parameters are obtained from either the Euler angles, or from a rotation about a unit vector that formulates the rotations into a single rotation about said unit vector. Given the Euler angles, the transformation matrix of a system is a combination of direction cosine matrices (DCM). The DCMs can be combined in various ways to represent a systems transformation matrix so long as a single representation is used for the duration of a problem.

$$A_1 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{26}$$

$$A_2 = \begin{bmatrix} \cos(\phi) & 0 & -\sin(\phi) \\ 0 & 1 & 0 \\ \sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \tag{27}$$

$$A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{bmatrix} \tag{28}$$

$$A = A_1 A_2 A_3$$

$$= \begin{bmatrix} cos(\phi)\,cos(\psi) - sin(\phi)\,cos(\theta)\,sin(\psi) & -cos(\phi)\,sin(\psi) - sin(\phi)\,cos(\theta)\,cos(\psi) & sin(\phi)\,sin(\theta) \\ sin(\phi)\,cos(\psi) + cos(\phi)\,cos(\theta)\,cos(\psi) & -sin(\phi)\,sin(\psi) + cos(\phi)\,cos(\theta)\,cos(\psi) & -sin(\theta)\,cos(\phi) \\ sin(\theta)\,sin(\psi) & sin(\theta)\,cos(\psi) & cos(\theta) \end{bmatrix} \tag{29}$$

The above Equations (26)-(28) depict three DCMs where equation (29) is one of many possible DCM combinations that create a transformation matrix for the system. With this transformation matrix, it can be seen that a singularity results when $sin(\theta) = 0$. A different combination of DCMs, or switching algorithm [11], can be implemented in an attempt to eliminate the singularity. The use of Euler parameters instead ensures no singularity will occur, but it introduces constraints that must be adhered to.

One approach [2] suggests that the Euler parameters replace the angles, angular velocity, and angular acceleration subject to a constraint.

$$\boldsymbol{p} = [e_0 \quad e_1 \quad e_2 \quad e_3]^T \tag{30}$$

$$\boldsymbol{C} = \boldsymbol{p}'\boldsymbol{p} - 1 \tag{31}$$

where $\boldsymbol{p}$ is a vector of Euler parameters. The formulation of a DAE using Euler parameters is then [2]:

$$4G^T J G\ddot{\boldsymbol{p}} = 2G^T \boldsymbol{n}' + 8\dot{G}^T J \dot{G}\boldsymbol{p} \tag{32}$$

where G is

$$G = \begin{bmatrix} -e_1 & e_0 & e_3 & -e_2 \\ -e_2 & -e_3 & e_0 & e_1 \\ -e_3 & e_2 & -e_1 & e_0 \end{bmatrix} \tag{33}$$

Another parameter, E, is then defined in order to calculate the transformation matrix as a function of Euler parameters.

$$E = \begin{bmatrix} -e_1 & e_0 & -e_3 & e_2 \\ -e_2 & e_3 & e_0 & -e_1 \\ -e_3 & -e_2 & e_1 & e_0 \end{bmatrix} \tag{34}$$

The transformation matrix is then defined as

$$A \equiv EG^T \tag{35}$$

The rotation of a system in global coordinates can now be found with a transformation matrix derived using either Euler angles or parameters, as long as it is consistent for the duration of a problem. Numerical methods may now be implemented on a formulated DAE to solve for translation and rotation of a constrained multibody dynamic system.

1.2.4 INDEX-3 DAE SOLUTION METHODS

Various numerical solution methods exist for DAEs. This paper will focus on using the projection method [12] and an alternate penalty method [16] on index-3 DAEs for comparison with the static penalty method in subsequent chapters.

The projection method has been selected due to its ease of implementation and solution accuracy. The projection method is derived from the DAE formulation in equation (21). Provided that the mass matrix, $\boldsymbol{M}$, is nonsingular, equation (21) can be rewritten using the LU decomposition [12]:

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{C}_{,q}^{T} \\ \boldsymbol{C}_{,q} & \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{C}_{,q} & -\boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{36}$$

Thus equation (21) can be rewritten as

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{C}_{,q} & -\boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{\gamma} \end{bmatrix} \tag{37}$$

defining two intermediate variables

$$\begin{bmatrix} \ddot{\boldsymbol{q}}^{*} \\ \boldsymbol{\lambda}^{*} \end{bmatrix} \equiv \begin{bmatrix} \boldsymbol{I} & \boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} \tag{38}$$

which can be illustrated explicitly as

$$\ddot{\boldsymbol{q}}^{*} = \ddot{\boldsymbol{q}} + \boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T}\boldsymbol{\lambda} \tag{39}$$

and

$$\boldsymbol{\lambda}^{*} = \boldsymbol{\lambda} \tag{40}$$

Equation (37) can then be rewritten as

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{C}_{,q} & -\boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}^{*} \\ \boldsymbol{\lambda}^{*} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{\gamma} \end{bmatrix} \tag{41}$$

From the first row of equation (41), $\boldsymbol{M}\ddot{\boldsymbol{q}}^{*} = \boldsymbol{F}$, so

$$\ddot{\boldsymbol{q}}^{*} = \boldsymbol{M}^{-1}\boldsymbol{F} \tag{42}$$

From the second row of equation (41)

$$\boldsymbol{C}_{,q}\ddot{\boldsymbol{q}}^{*} - \boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T}\boldsymbol{\lambda}^{*} = \boldsymbol{\gamma} \tag{43}$$

which, upon solving for $\boldsymbol{\lambda}^{*}$ and substituting the result of equation (42), yields

$$\boldsymbol{\lambda}^{*} = \left(\boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T}\right)^{-1}\left(\boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{F} + \boldsymbol{\gamma}\right) \tag{44}$$

Now, from equation (39),

$$\begin{aligned} \therefore \ddot{\boldsymbol{q}} &= \ddot{\boldsymbol{q}}^{*} - \boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T}\boldsymbol{\lambda}^{*} \\ &= \boldsymbol{M}^{-1}\boldsymbol{F} - \boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T}\boldsymbol{\lambda} \end{aligned} \tag{45}$$

Thus, the first row of equation (41) is,

$$\boldsymbol{M}\ddot{\boldsymbol{q}} = \boldsymbol{F} - \boldsymbol{C}_{,q}^{T}\boldsymbol{\lambda} \tag{46}$$

However, at this point, based upon equation (44), one can calculate $\boldsymbol{\lambda}$

$$\boldsymbol{\lambda} = \left(\boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{C}_{,q}^{T}\right)^{-1}\left(\boldsymbol{C}_{,q}\boldsymbol{M}^{-1}\boldsymbol{F} + \boldsymbol{\gamma}\right) \tag{47}$$

One may also substitute the values for both $\ddot{\boldsymbol{q}}^{*}$ and $\boldsymbol{\lambda}^{*}$ from equations (42) and (44) to obtain

$$\begin{aligned}\ddot{\boldsymbol{q}} &= \boldsymbol{M}^{-1}\boldsymbol{F} - \boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{F} - \boldsymbol{\gamma}\right) \\ &= \boldsymbol{M}^{-1}\boldsymbol{F} - \boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{F} \\ &+ \boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{\gamma} \\ &= \boldsymbol{M}^{-1}\left(\boldsymbol{I} - \boldsymbol{C}_{,\boldsymbol{q}}^{T}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\right)\boldsymbol{F} + \boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{\gamma} \\ &= \boldsymbol{M}^{-1}\left(\boldsymbol{F} - \boldsymbol{C}_{,\boldsymbol{q}}^{T}\left[\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{F} - \left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{\gamma}\right]\right) \\ &= \boldsymbol{M}^{-1}\left(\boldsymbol{P}\boldsymbol{F} - \boldsymbol{C}_{,\boldsymbol{q}}^{T}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{\gamma}\right)\end{aligned} \tag{48}$$

where

$$\boldsymbol{P} \equiv \boldsymbol{I} - \boldsymbol{C}_{,\boldsymbol{q}}^{T}\left(\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1}\boldsymbol{C}_{,\boldsymbol{q}}^{T}\right)^{-1}\boldsymbol{C}_{,\boldsymbol{q}}\boldsymbol{M}^{-1} \tag{49}$$

The projection method here only satisfies the constraints specified for $\ddot{\boldsymbol{q}}$. Hidden constraints must also be adhered to for the displacements and velocities. These hidden constraints are the same specified in equations (23) and (24). The complexity of the projection method should be noted here, as it is equations (48) and (49) that must be implemented within a computer program.

An alternate penalty method derived in [16] is selected to show that other penalty methods exist in literature and provide a more direct comparison with the static penalty method. For an index-3 DAE, the alternate penalty method treats $\boldsymbol{q}$ as unknown. In the subsequent chapter, it is seen that for the static penalty method $\ddot{\boldsymbol{q}}$ is unknown. Using virtual work and estimating Lagrange multipliers with the penalty method, [16] depicts

$$\left(M\ddot{\boldsymbol{q}} + \boldsymbol{C}_{,\boldsymbol{q}}^{T}\mu\boldsymbol{C} - \boldsymbol{F}\right)^{T}\delta\boldsymbol{q} = \boldsymbol{0} \tag{50}$$

where $\delta\boldsymbol{q}$ is an arbitrary constant. This results in

$$M\ddot{\boldsymbol{q}} + \boldsymbol{C}_{,\boldsymbol{q}}^{T}\mu\boldsymbol{C} - \boldsymbol{F} = \boldsymbol{0} \tag{51}$$

which can be approximated as

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{F} - \boldsymbol{C}_{,\boldsymbol{q}}^{T}\mu\boldsymbol{C}) \tag{52}$$

where μ is the penalty coefficient.

The illustrated projection method (equations 48 and 49) and an alternate penalty method (equation 52) can now be utilized to numerically solve an index-3 DAE. Both methods are solved for at each time step to ensure the EOMs adhere to the body and hidden constraints. The updated variables may then be integrated numerically for the resulting ODEs using methods described in Section 1.2.2.

1.2.5 INDEX-2 DAE SOLUTION METHODS

Although the primary focus of this thesis is on Index-3 DAEs, a solution method for an Index-2 DAE is presented for completeness. The static penalty method derived in Chapter 2 will also contain formulations for both an Index-3 and Index-2 DAE.

In order to formulate an Index-2 DAE, a set of first order ODEs are observed as

$$M\dot{\boldsymbol{y}} = \boldsymbol{F}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{y}$$

These ODEs can be combined into a matrix illustrated as equation (53) and are subject to a constraint matrix, as depicted in equation (54).

$$\begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix}\begin{bmatrix} \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t) \\ y \end{bmatrix} \tag{53}$$

$$\begin{bmatrix} \boldsymbol{C}_{,\boldsymbol{q}} & 0 \\ 0 & \boldsymbol{C}_{,\boldsymbol{q}} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{q}} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{\gamma}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t) \\ \boldsymbol{C}_{,t} \end{bmatrix} \tag{54}$$

Upon application of Lagrange multipliers and combination of equations (53) and (54), the Index-2 DAE is represented as

$$\begin{bmatrix} M & 0 & (\boldsymbol{C}_{,\boldsymbol{q}})^T & 0 \\ 0 & I & 0 & (\boldsymbol{C}_{,\boldsymbol{q}})^T \\ \boldsymbol{C}_{,\boldsymbol{q}} & 0 & 0 & 0 \\ 0 & \boldsymbol{C}_{,\boldsymbol{q}} & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{q}} \\ \boldsymbol{\lambda}_{\boldsymbol{y}} \\ \boldsymbol{\lambda}_{\boldsymbol{g}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{y} \\ -\boldsymbol{\gamma} \\ -\boldsymbol{C}_{,t} \end{bmatrix} \tag{55}$$

From [16], the formulation for an alternative Index-3 DAE penalty method was illustrated in the previous section. This reference also contains the derivation for an alternative Index-2 DAE. Applying virtual work to equation (53) results in equation (56) where the variations must satisfy equation (57).

$$\left(\begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{q}} \end{bmatrix} - \begin{bmatrix} \boldsymbol{F} \\ y \end{bmatrix} \right)^{\boldsymbol{T}} \begin{bmatrix} \boldsymbol{\delta y} \\ \boldsymbol{\delta q} \end{bmatrix} = \mathbf{0} \tag{56}$$

$$\begin{bmatrix} \boldsymbol{C}_{,\boldsymbol{q}} & 0 \\ 0 & \boldsymbol{C}_{,\boldsymbol{q}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta y} \\ \boldsymbol{\delta q} \end{bmatrix} = \mathbf{0} \tag{57}$$

Combining equations (56) and (57) and introducing the alternate penalty factor as an estimate for the Lagrange multipliers, the final alternate Index-2 penalty formulation is [16]

$$\begin{bmatrix} M & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{y}} \\ \dot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t) - \boldsymbol{C}_{,\boldsymbol{q}}^T \mu \left(\boldsymbol{C}_{,\boldsymbol{q}} \dot{\boldsymbol{q}} + \boldsymbol{C}_{,\boldsymbol{q}} \right) \\ \boldsymbol{y} - \boldsymbol{C}_{,\boldsymbol{q}}^T \mu \boldsymbol{C} \end{bmatrix} \tag{58}$$

Equation (58) can now be utilized to numerically solve an Index-2 DAE. This method will be utilized with an example problem presented in Chapter 2 as a comparison with the static Index-3 penalty formulation.

1.2.6 FINITE ELEMENT STATIC PENALTY METHOD

In statics, analysis is performed on objects that are in static equilibrium. Similarly to dynamic problems, there are various methods for solving constrained static problems. One such method is a static penalty approach [13] that handles boundary conditions for a wide array of static problems. The appeal of utilizing this penalty method stems from its ease of application, computational suitability, and accuracy.

In statics, the displacements are of interest for analysis of structures. It is typical to constrain a static system to a wall where no displacement is allowed. This connection can be modelled as a spring with a large stiffness. The large spring stiffness will then act as a penalty for variables interacting with this point on the wall. From [13], the constraint can be applied directly at the node interacting with a wall.

$$Q_1 = a_1 \tag{59}$$

The strain energy in the spring of large stiffness is

$$U = \frac{1}{2} C(Q_1 - a_1)^2 \tag{60}$$

Making the potential energy of the system

$$\Pi = \frac{1}{2} Q^T K Q + \frac{1}{2} C(Q_1 - a_1)^2 - Q^T F \tag{61}$$

In order to minimize the potential energy, the derivative of Π is set equal to zero resulting in

$$\begin{bmatrix} K_{11}+C & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & & \vdots \\ \vdots & & \ddots & \\ K_{n1} & \cdots & & K_{nn} \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_n \end{bmatrix} = \begin{bmatrix} F_1 + Ca_1 \\ F_2 \\ \vdots \\ F_n \end{bmatrix} \tag{62}$$

where

$$C = max\left|K_{ij}\right| \times 10^4 \tag{63}$$

Equation (62) illustrates a system constrained at a single node. The constraint penalty (C) is added to the diagonal of the stiffness matrix that represents the node being restricted. Concurrently, the force at the constrained point is then summed with the constraint and displacement product. A single point constraint can be expanded to capture multipoint constraints. In order to implement multipoint constraints, the single point constraints are first applied as

$$\boldsymbol{C_C} = C \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1n} \\ & C_{22} & & \vdots \\ \vdots & & \ddots & \\ & & & C_{nn} \end{bmatrix} \tag{64}$$

Where C_{ij} = 0 if i does not equal j, and C_{ij} = 1 if i is equal to j. Boundary conditions of the following form of equation (65) may then be utilized to formulate a boundary constraint vector in equation (66) for n number of boundary conditions.

$$\beta_{ni}Q_{ni} + \beta_{nj}Q_{nj} = \beta_0 \tag{65}$$

$$\boldsymbol{B} = \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1Q} \\ \beta_{21} & \beta_{22} & & \vdots \\ \vdots & & \ddots & \\ \beta_{n1} & \cdots & & \beta_{nQ} \end{bmatrix} \tag{66}$$

The constrained stiffness matrix can then be formulated as

$$\boldsymbol{K_c} = \boldsymbol{K} + C(\boldsymbol{B}^T\boldsymbol{B} + \boldsymbol{C_C}) \tag{67}$$

Figure (1) illustrates an example of a static multipoint constraint problem [13]. Body 1 is defined with a cross sectional area of 1200 mm^2, modulus of elasticity of 200*10^3 N/mm^2, and length of 4.5 m. Body 2 is defined with a cross sectional area of 900 mm^2, modulus of elasticity of 70*10^3 N/mm^2, and length of 3 m. The load placed on the end of the beam is $30*10^3$ N. Application of the above penalty method using a MATLAB® computer script depicted in Appendix A.1 results in calculated values for the evaluated displacement parameters.

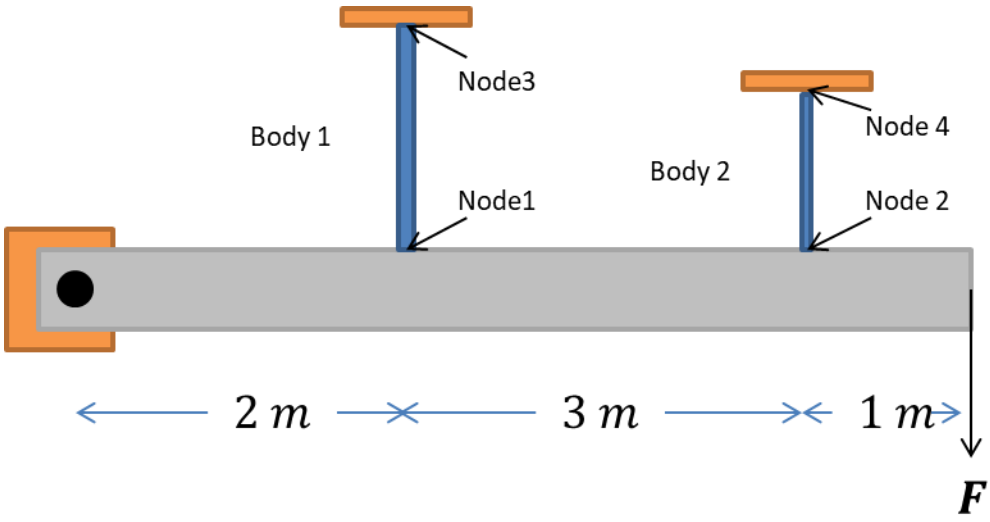

Figure 1: Static Multipoint Constraint

$$\boldsymbol{K} = \begin{bmatrix} 5.333 & 0 & -5.333 & 0 & 0 \\ 0 & 2.1 & 0 & -2.1 & 0 \\ -5.333 & 0 & 5.333 & 0 & 0 \\ 0 & -2.1 & 0 & 2.1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} 10^4$$

$$\boldsymbol{F} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 30 \end{bmatrix} 10^3$$

Displacements at Node 1 and Node 2 are constrained by the rigid bar rotation. This results in each node displacement equaling the displacement at the end of the bar (Q_5) multiplied by the ratio of the node length to total bar length. The boundary conditions can then be defined as

$$Q_1 - \frac{2}{6} Q_5 = 0$$

$$Q_2 - \frac{5}{6} Q_5 = 0$$

Applying equations (66), (63), and (64), the constraining equations are

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & -\frac{2}{6} \\ 0 & 1 & 0 & 0 & -\frac{5}{6} \end{bmatrix}$$

$$C = 5.333 \times 10^8$$

$$\boldsymbol{C_C} = C \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Substituting these values into equation (67) calculates the constrained stiffness matrix as

$$\boldsymbol{K_c} = \begin{bmatrix} 5.3339 & 0 & -0.0005 & 0 & -1.7778 \\ 0 & 5.3335 & 0 & -0.0002 & -4.4444 \\ -0.0005 & 0 & 5.3339 & 0 & 0 \\ 0 & -0.0002 & 0 & 5.3335 & 0 \\ -1.7778 & -4.4444 & 0 & 0 & 4.2963 \end{bmatrix} 10^8$$

The displacements are then calculated by replacing the stiffness matrix in equation (62) with the constrained stiffness matrix to find

$$\boldsymbol{Q} = \begin{bmatrix} 0.4876 \\ 1.2191 \\ 0 \\ 0 \\ 1.4629 \end{bmatrix} [mm]$$

It is seen that by applying the penalty method to this problem, the displacements are significantly diminished at the constraint points. Comparatively, displacements in the above static scenario have the potential to correlate with the displacements specified in equation (21) of the multibody dynamic DAE. It is the objective for the remainder of this thesis to quantitatively extend the static penalty method for use with constrained multibody dynamic DAEs.

CHAPTER 2. PENALTY METHOD IMPLEMENTATION

In this chapter, the formulation of a linear multipoint static penalty method from Section 1.2.6 will be modified to encompass the EOMs of a constrained multibody dynamic system. A stabilization method derived in [17] and illustrated as equation (25) is then incorporated with the penalty method as an alternative for solving the hidden constraints. A brief overview of coding software and applicable functions in MATLAB® will follow. Code will apply an algorithm to subsequent example problems, comparing the static penalty method of Section 1.2.6 to the projection method from equations (47) to (58) and an alternate penalty method from equation (51). Example problems presented for examining the static penalty method are a simple pendulum, a Scotch mechanism, and a slider-crank mechanism.

2.1 EXTENDED STATIC PENALTY METHOD DERIVATION

The static penalty method applies a constraint (modelled as a spring with large stiffness) to the stiffness matrix and force vector of equation (62). The constrained stiffness matrix of equation (67) replaces the K matrix for multipoint constraints. Comparing equations (6) and (62) it is observed that direct correlations exist between the mass and stiffness matrix, displacement vectors, and force vectors when formulating the potential energy of a system. The penalized quadratic constraint of the static system must then be formulated in terms of multibody dynamics. This constraint for the static system was formulated from the fact that displacement at an attached point must equal the attachment displacement. The linear algebraic equation of $\ddot{\boldsymbol{q}}$ can then be evaluated from equation (18) by setting $\boldsymbol{\lambda}$ equal to zero resulting in equation (68) which must satisfy equation (69).

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} - \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = \boldsymbol{0} \tag{68}$$

$$\boldsymbol{C}_{,\boldsymbol{q}}\ddot{\boldsymbol{q}} + \boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = \boldsymbol{0} \tag{69}$$

Static variables of equation (61) can then be replaced by their multibody dynamic counterparts, where the displacement constraint of equation (59) is replaced by the linear acceleration constraint above (equation (69)). Reformulating equation (69) with dynamics variables for a constrained multibody system results in equation (70) where $\ddot{\boldsymbol{q}}$ is considered as the unknown.

$$\Pi = \frac{1}{2}\ddot{\boldsymbol{q}}^{\boldsymbol{T}}\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \frac{1}{2}\mu(\boldsymbol{C}_{,\boldsymbol{q}}\ddot{\boldsymbol{q}} + \boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t))^{T}(\boldsymbol{C}_{,\boldsymbol{q}}\ddot{\boldsymbol{q}} + \boldsymbol{\gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)) - \ddot{\boldsymbol{q}}^{T}\boldsymbol{F} \tag{70}$$

Differentiating equation (70) with respect to $\ddot{\boldsymbol{q}}$ provides a solution formulated as equation (71).

$$\left(\boldsymbol{M} + \mu\boldsymbol{C}_{,\boldsymbol{q}}^{T}\boldsymbol{C}_{,\boldsymbol{q}}\right)\ddot{\boldsymbol{q}} = \boldsymbol{F} - \mu\boldsymbol{C}_{,\boldsymbol{q}}^{T}\gamma \tag{71}$$

Comparing equation (71) to equation (18) shows that the Lagrangian multiplier ($\boldsymbol{\lambda}$) has been successfully eliminated. This equation can now provide solutions for acceleration components of the constrained multibody dynamic system through direct solutions at each time step of the problem. Additional hidden constraints described in section 1.2.3 must also be adhered to as integrations are computed at each time step for $\dot{\boldsymbol{q}}$ and $\boldsymbol{q}$**.** Hidden constraints for the system are depicted in equations (23) and (24). The Baumgarte stabilization may also be applied to the DAE solved for in equation (71). If the Baumgarte stabilization from equation (25) is utilized, the hidden constraints are implicitly solved for at each time step and equation (71) is now equation (72).

$$\left(\boldsymbol{M} + \mu \boldsymbol{C}_{,\boldsymbol{q}}^{T} \boldsymbol{C}_{,\boldsymbol{q}}\right)\ddot{\boldsymbol{q}} = \boldsymbol{F} - \mu \boldsymbol{C}_{,\boldsymbol{q}}^{T}(-\gamma - \mathbf{2}\alpha\dot{\boldsymbol{C}} - \zeta \boldsymbol{C}) \tag{72}$$

The formulation of equations (71) and (72) depict extensions of the static penalty method for use with constrained multibody dynamics. Following the static penalty formulation, an estimate for the dynamic penalty factor can be calculated using the maximum mass component time 10^5. In order to implement these formulations with practical problems, an algorithm was developed. This algorithm allows for computational modelling of constrained multibody dynamic problems using the static penalty method. In order to computationally implement this algorithm, MATLAB® by mathworks was utilized as the base coding platform. Computations were run on an Intel® Core™ i5-2410M CPU @ 2.30 GHz with 8.0 GB of RAM. The MATLAB® version utilized is R2014a.

2.2 SOFTWARE AND FUNCTIONS

MATLAB® is a computer coding platform utilized academically and industrially by engineers and scientists as a highly robust data analysis tool [14]. MATLAB® will be utilized frequently throughout the remainder of this paper for calculations and models of constrained multibody dynamics. All MATLAB® scripts created in support of this research are illustrated in Appendix A.

When using MATLAB®, a suite of functions are available to users that create shortcuts for implementing algorithms. One such function that is widely used in this research is “ODE23”, a three stage, third order Runge-Kutta ordinary differential solver [15]. In order to solve a DAE using MATLAB®, the following algorithm was implemented:

1. Define the initial conditions at time equal to zero.
2. Define the time span and time step for integrations.
3. Reference initial parameters and time step for ODE 23.
4. ODE23 then refers to a function containing the system constraints and EOMs.
5. Within the ODE23 function, the body constraints and hidden constraints are solved simultaneously.
6. The acceleration variables are then solved for using a specific DAE solution method.
7. ODE23 steps through the timespan at the specified time step.
8. Numerical parameters are returned to the original script.
9. Parameters are then utilized for post-processing.

This algorithm will be implemented for subsequent example problems. MATLAB® scripts are illustrated in Appendix A.

2.3 EFFECTS OF THE PENALTY FACTOR

A penalty factor of $\mu = 10^5$ is utilized for the following example problems. This penalty factor was chosen due to its implementation; it resulted in static penalty solutions that are comparable to projection method solutions in both computational time and solution accuracy for the initial simple pendulum example. An analysis of varying μ from 10 to 10^8 is depicted in Figures (2)-(4) for static penalty without stabilization, static penalty with stabilization, and alternate penalty methods, respectively. This analysis was done using the simple pendulum example due to the use of a direct numerical approach as a baseline for all other solution methods. In this case the maximum mass component is one. Computational times for each

method over the span of penalty factors is depicted in Table (1). Small changes in both computational time and solution accuracy are observed for the static penalty with stabilization. Large deviations in solution accuracy and small variation of computation time is observed for the static penalty without stabilization. Large variations in both solution accuracy and computational time are observed for the alternate penalty method. In all three penalty methods, it is obvious that solutions are more accurate as penalty factors increase. Since the stabilization handles hidden constraints more precisely, there is significantly less error when the penalty factor is applied. This results in solutions that are more dependent on the Baumgarte weighting when analyzing accuracy and computational time.

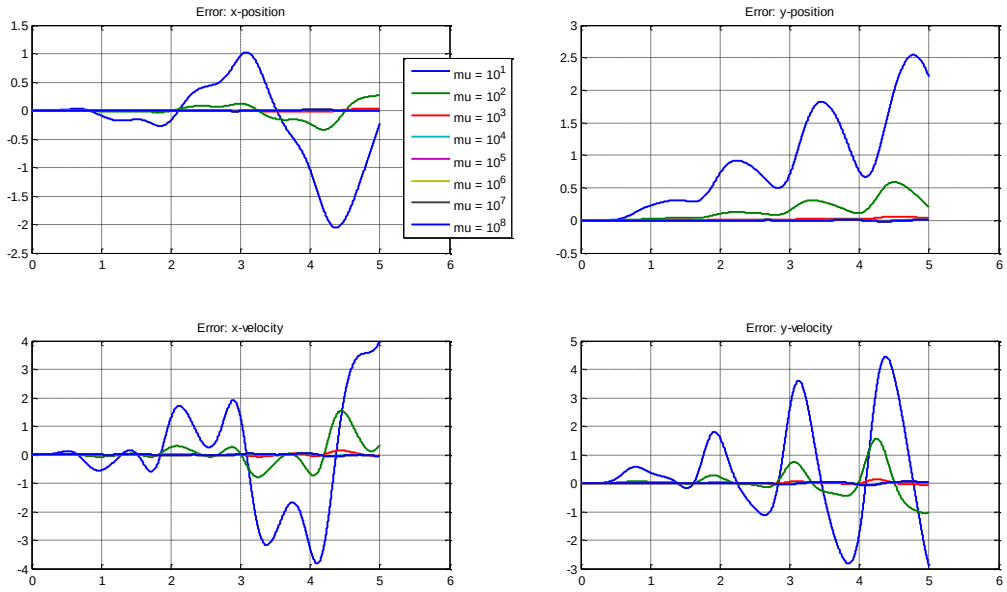

Figure 2: Static Penalty without Stabilization

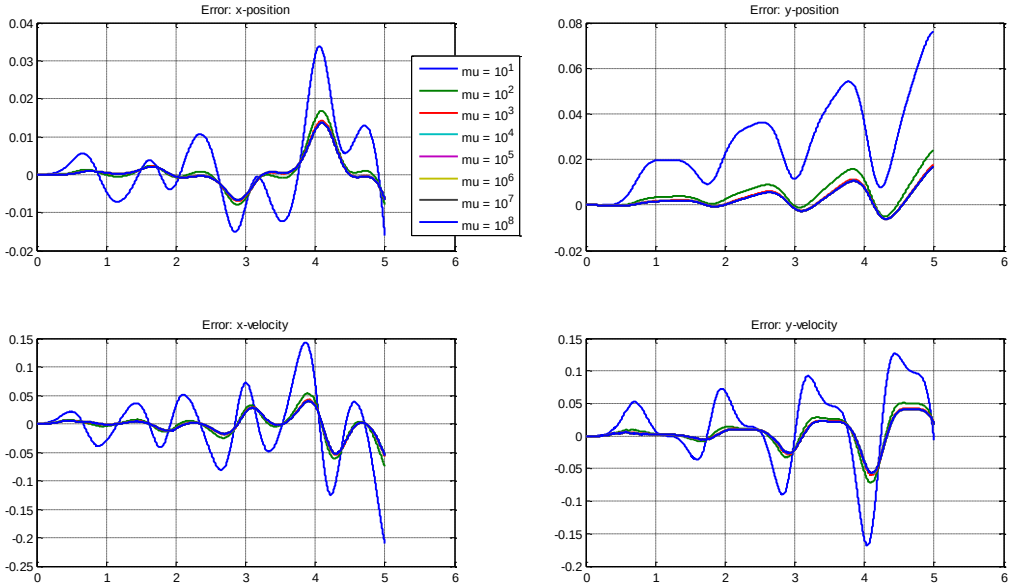

Figure 3: Static Penalty with Stabilization

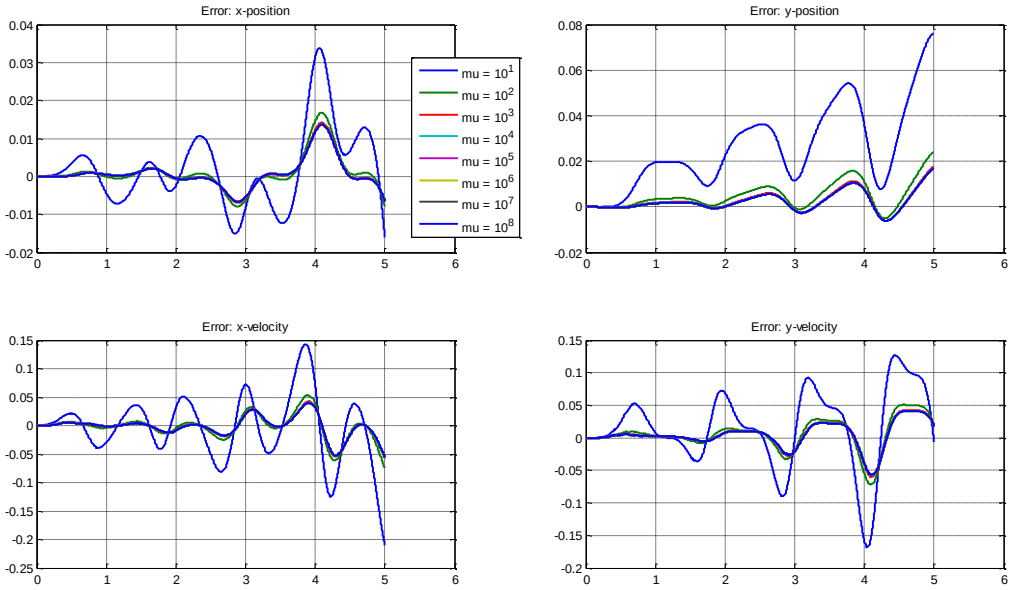

Figure 4: Alternate Penalty

Table 1: Computational Time Over Varying Penalty Factor

μ	Static Penalty*	Static Penalty, Baumgarte**	Goicolea and Orden
10	1.1112	0.9679	0.8816
10^2	0.9061	0.9441	0.9254
10^3	0.9291	0.9461	0.9874
10^4	0.9457	0.9825	1.1233
10^5	0.9365	1.0045	1.6363
10^6	0.9418	0.9546	2.7658
10^7	0.9555	0.9673	6.8599
10^8	0.9373	0.9409	19.3364
*Projection = 1.0887 **Projection Baumgarte = 0.9484			

[seconds]

It is also observed that as the mass becomes significant when compared with the penalty factor, additional error is introduced. In order to mitigate this, the penalty factor should be five orders of magnitude higher than the largest factor within the mass matrix. This will result in solutions that are comparable in error to those presented in Figure 3.

2.4 EXAMPLE PROBLEMS

The following example problems were chosen as a means to verify that application of a static penalty method was feasible. Problems of increasing complexity are then used to evaluate the static penalty method as well as its computational efficiency. The first example problem depicts a simple pendulum strictly meant for validation purposes. This problem was selected due to its simplicity for evaluating the effectiveness of implementing the static penalty method as compared with a projection and an alternate penalty method. The use of Baumgarte stabilization [17] is also assessed using this problem. For completeness, the simple pendulum is also examined using an Index-2 DAE. The next example problem implements the static penalty method with a Scotch mechanism. The Scotch mechanism is a relatively simple problem, but it

demonstrates applicability towards a constrained multibody dynamic system. A slider-crank mechanism is then observed in the final problem. Slider-crank mechanism formulations can be extrapolated to encompass a variety of other constrained multibody systems, making this an ideal problem for testing the applicability of the derived penalty method. For both the Scotch and slider-crank mechanisms, the static penalty method is compared against projection and alternate penalty method solutions.

2.4.1 SIMPLE PENDULUM

The single body problem chosen for this research is that of a simple pendulum, as illustrated in Figure (5). The mass of the system is concentrated on a point length (L) from the fixed origin. Gravity is acting uniformly over the system in the positive y direction with no external torques or forces. Damping is not present in the system; therefore, results are expected to represent a simple harmonic oscillator.

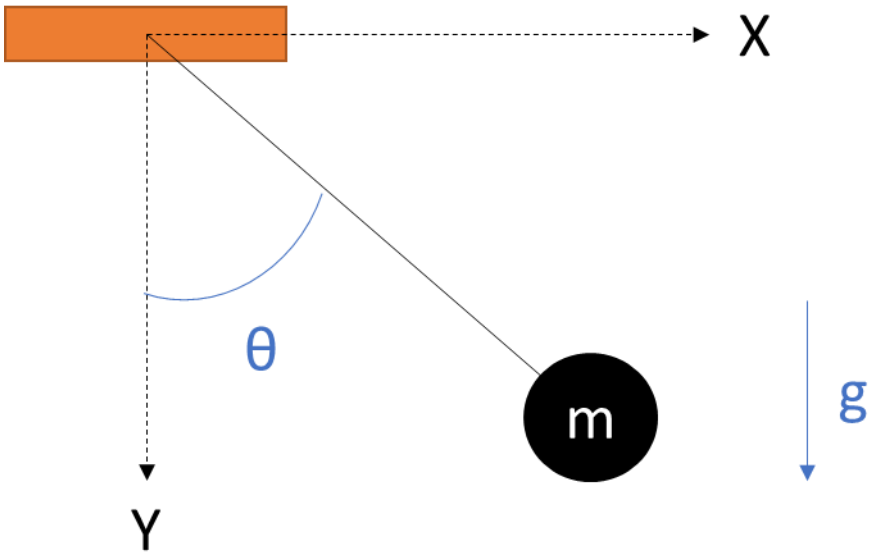

Figure 5: Simple Pendulum

Formulations for the equations of motion are then derived from Equations (4) and (5) as

$$\boldsymbol{M} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix}$$

$$\boldsymbol{F} = \begin{bmatrix} 0 \\ mg \\ 0 \end{bmatrix}$$

Constraints in Cartesian coordinates for the simple pendulum must ensure that the mass stays on a circular path and does not deviate in the z direction. The holonomic constraint from Equation (19) is then formulated as

$$\boldsymbol{C} = \begin{bmatrix} x^2 + y^2 - 1 \\ z \end{bmatrix}$$

The Jacobian of the $\boldsymbol{C}$ matrix is then found using equation (20).

$$\boldsymbol{C}_{,\boldsymbol{q}} = \begin{bmatrix} 2x & 2y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Equation (22) is then used to calculate $\boldsymbol{\gamma}$:

$$\boldsymbol{\gamma} = -\begin{bmatrix} 2\dot{x}^2 & 2\dot{y}^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

These variables are then substituted into Equation (21) to formulate the DAE of the constrained simple pendulum, where

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}$$

Initial conditions for the simple pendulum are listed in Table 2. Initial conditions and variables are then substituted into the MATLAB® algorithm described in Section 2.2. The MATLAB® script for this problem is depicted in Appendix A.2 and calls ODE23 functions seen in Appendix A.2.a, A.2.b, A.2.c, A.2.d, A.2.e, A.2.f, and A.2.g.

Table 2: Simple Pendulum Initial Conditions

g [m/s^2]	m [kg]	x_0 [m]	y_0 [m]	z_0 [m]	$\dot{x}_0$ [m/s]	$\dot{y}_0$ [m/s]	$\dot{z}_0$ [m/s]
9.81	1	1	0	0	0	0	0
t_{init} [s]	dt [s]	t_{final} [s]	Constraint Tolerance				
0	0.0001	5	1^{-10}				

Figure (6) illustrates the pendulum motion as a result of the penalty method equation (71). For this case, a penalty factor (μ) of 10^5 is implemented on the constraints. The simulation is run for a period of five seconds. Figure (7) depicts the error between the static penalty solution and a direct numerical integration of equation (72) using ODE23.

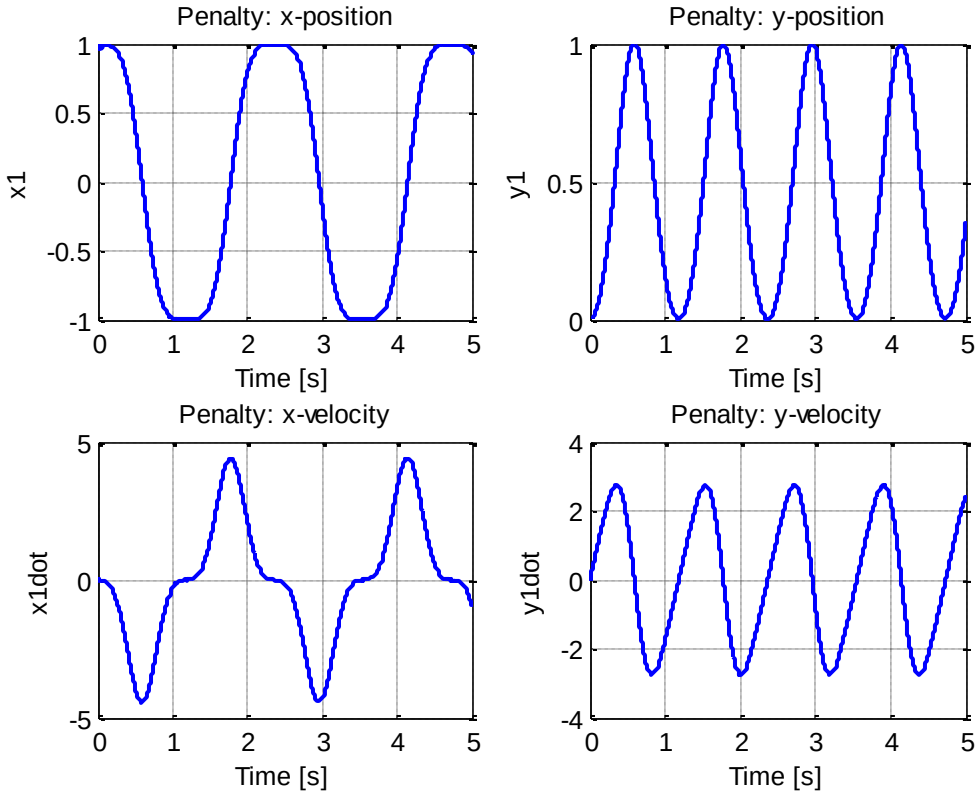

Figure 6: Simple Pendulum Motion Plots

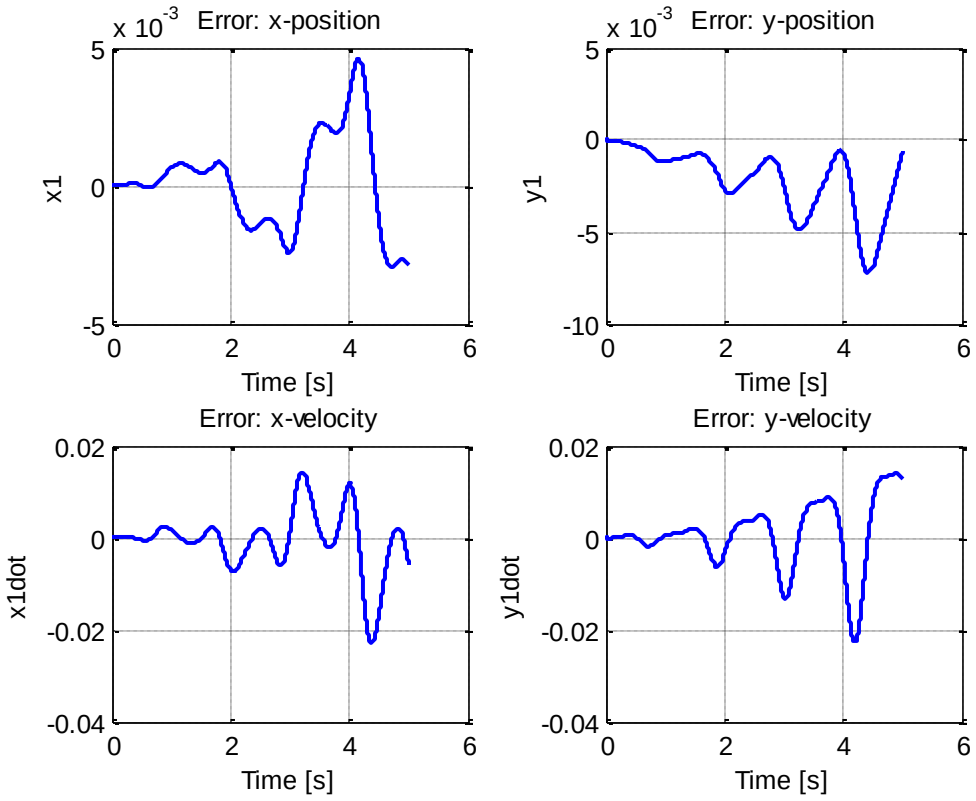

Figure 7: Simple Pendulum Penalty Error

Solution accuracy is compared against a direct numerical integration of a simple pendulum illustrated as equation (73). ODE23 was also utilized for this direct numerical approach and is depicted in Appendix A.2.g. One hundred computations were performed for each method in order to calculate a reasonable mean of the computational time. The static penalty method averaged a computational time of 0.9365 seconds, whereas the direct numerical approach has a mean computational time of 0.8391 seconds over 100 computations.

$$\ddot{\theta} = -\frac{g}{L}\sin(\theta) \tag{73}$$

Since the direct numerical approach provides an output of θ and $\dot{\theta}$, position and velocity are calculated using equation (74) through (77).

$$x = Lsin(\theta) \tag{74}$$

$$y = Lcos(\theta) \tag{75}$$

$$\dot{x} = L\omega cos(\theta) \tag{76}$$

$$\dot{y} = -L\omega sin(\theta) \tag{77}$$

In order to determine if the error between the static penalty method and direct numerical approach is acceptable, the static penalty method solution is compared with the projection method from equation (48) in Figure (8).

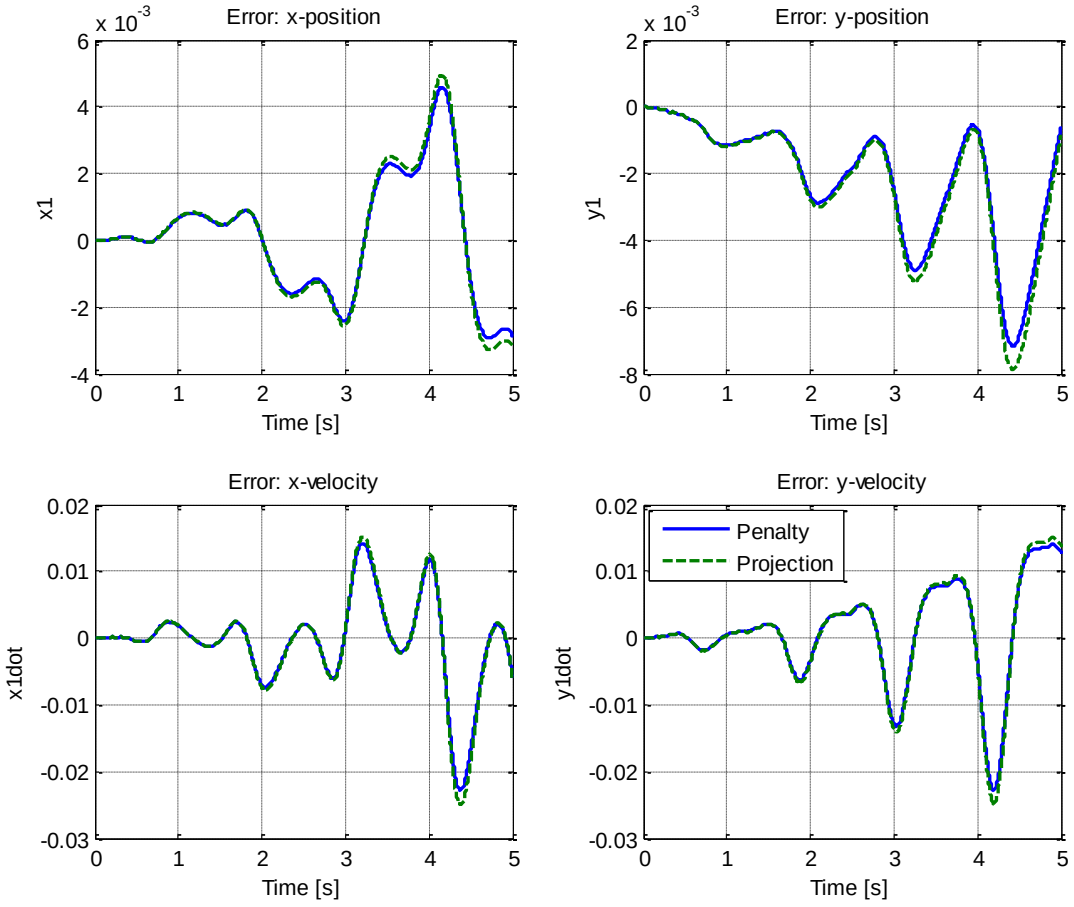

Figure 8: Pendulum Penalty vs. Projection

From Figure (8), it is observed that with a penalty factor of 10^5, there is no discernable difference between the two methods. A mean computational time for the projection method was found to be 1.0887 seconds measured over 100 computations. Comparatively, both computational times are reasonably low. The static penalty method was therefore verified as a reasonable approach for solving dynamic systems.

Baumgarte stabilization is then applied to both the static penalty and projection methods. This stabilization method performs the displacement and velocity corrections

implicitly, reducing the amount of code required for each function. The motion plots for both methods containing the Baumgarte stabilization are not discernable from Figure (6). The difference between the static penalty and projection methods with and without the Baumgarte stabilization are represented in Figures (9) and (10) respectively. The static penalty method with stabilization averaged 1.0045 seconds over 100 computations. The projection averaged 0.9484 seconds. Weighting coefficients of α = 20 and β = 4.4721 are utilized per [17]. A penalty factor of $\mu = 10^5$ is kept.

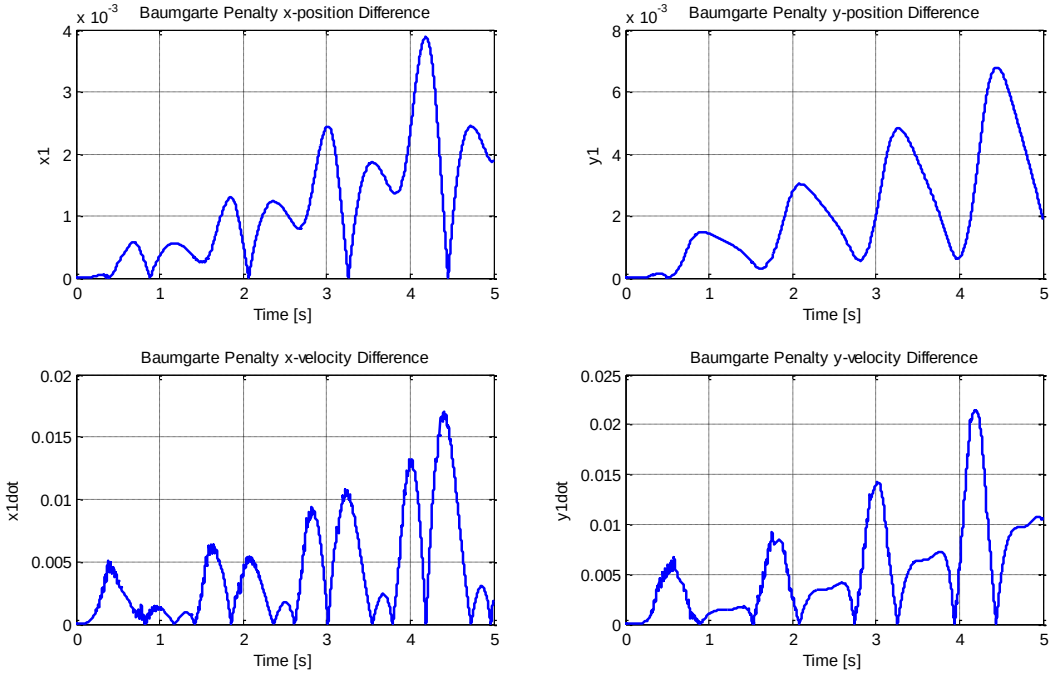

Figure 9: Static Penalty Baumgarte Comparison

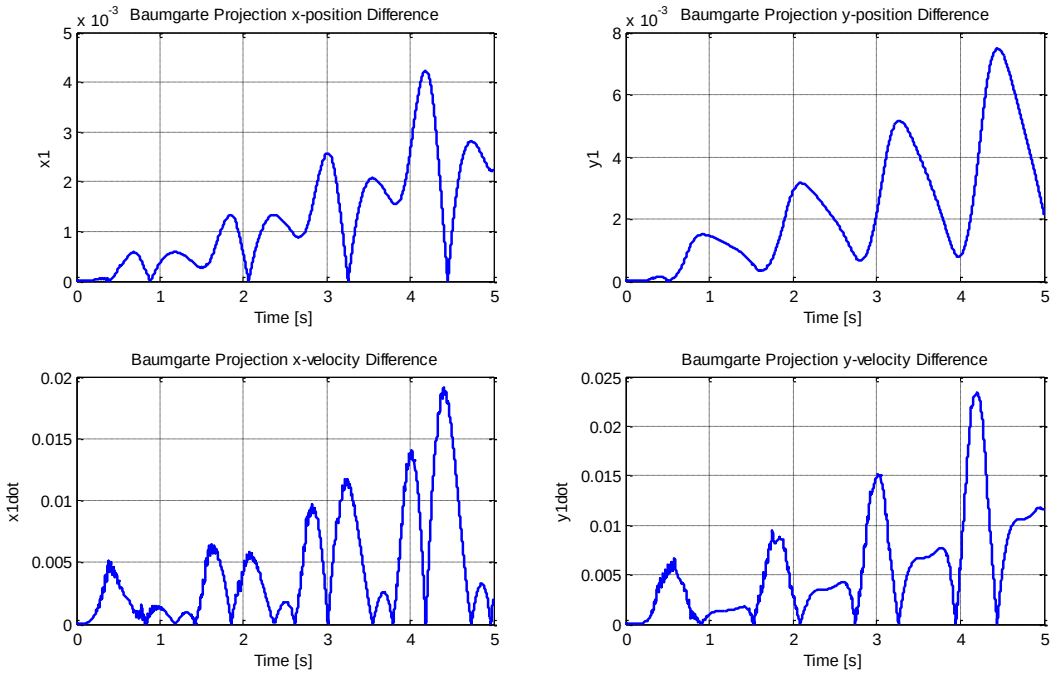

Figure 10: Projection Baumgarte Comparison

Comparing the static penalty method errors with and without Baumgarte stabilization demonstrates a slight increase in accuracy when stabilization is used. This comparison is illustrated in Figure (11). From these observations, the static penalty and projection methods utilized for subsequent example problems will have hidden constraints enforced using Baumgarte stabilization, as significantly less coding is required.

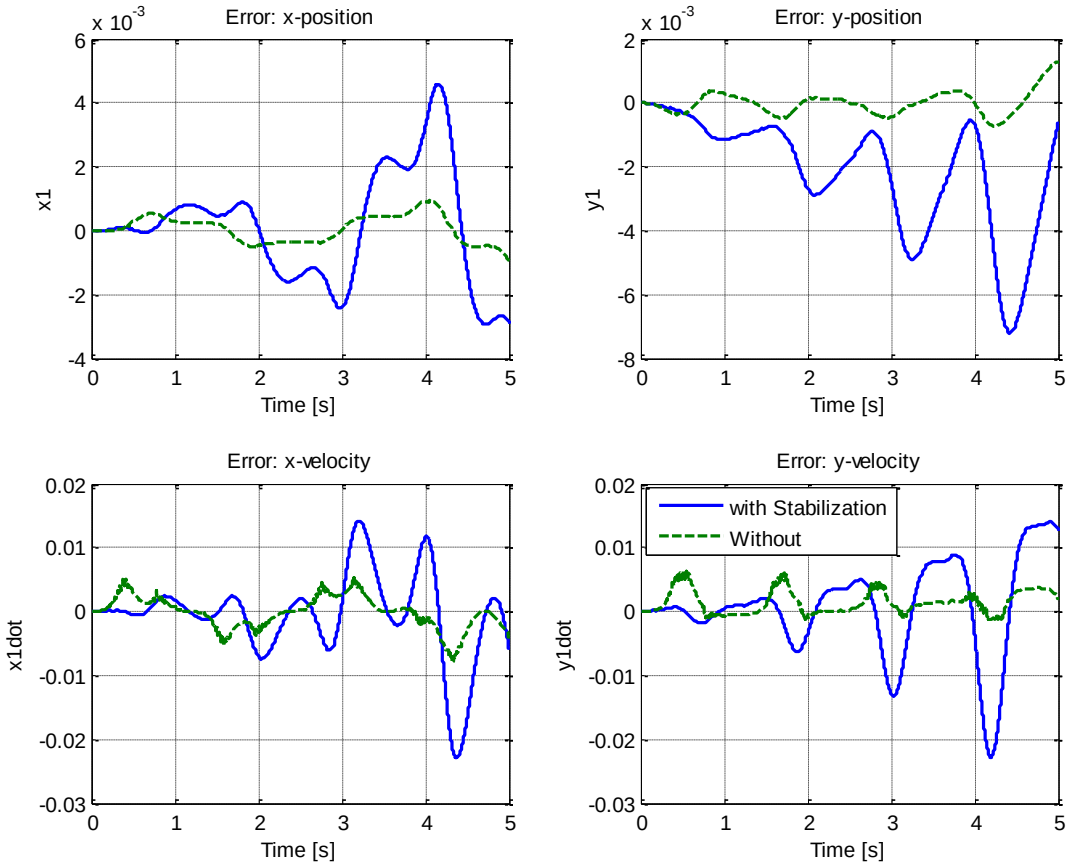

Figure 11: Static Penalty Error Stabilization Comparison

An alternate penalty method depicted as equation (52) is then utilized to provide an additional comparator with the static penalty method. The alternate penalty method from [16] also demonstrates that additional methods have been derived that implement a penalty factor on constraints to solve dynamic problems. Motion solutions using the alternate penalty method are not discernable from Figure (6). Error, as compared with the direct numerical approach, is slightly more accurate than that of the previous methods but contains much more noise in the results. The noise is especially dominant in the velocity error comparisons. Computational time

for the alternate method is found to be 1.6363 seconds averaged over 100 runs. The alternate error is depicted with the static error in Figure (12).

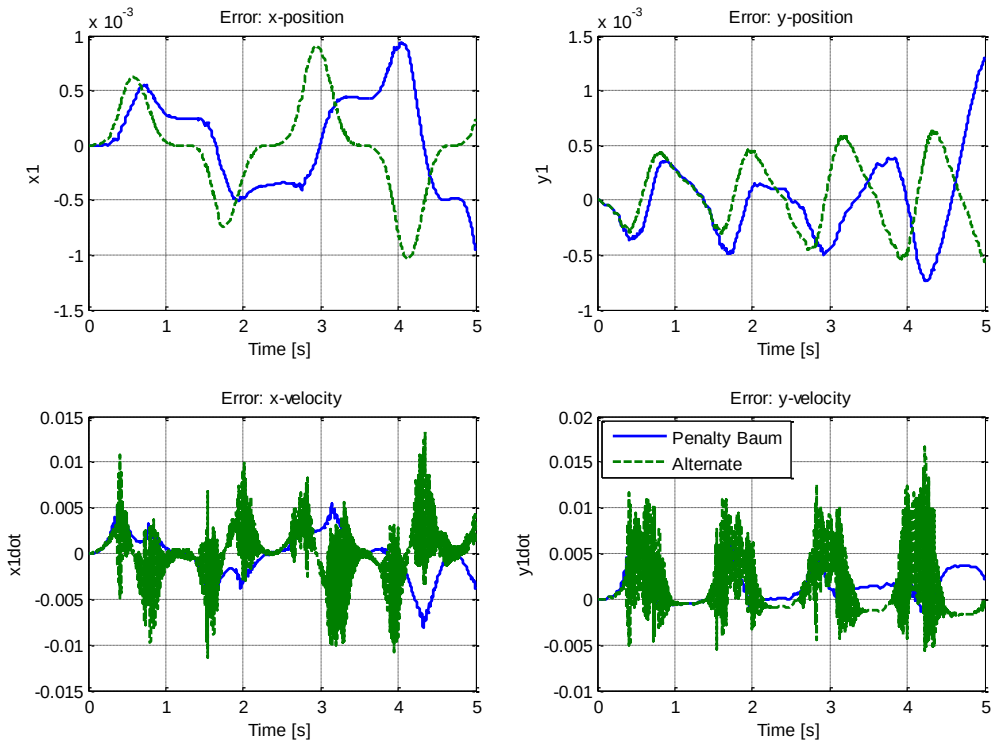

Figure 12: Alternate Penalty Error

For completeness, the static penalty method is compared with an Index-2 penalty approach derived in [17] and represented as equation (58). Motion solutions using the Index-2 penalty method are not discernable from Figure (6). Error as compared with the direct numerical approach is slightly less accurate than that of the previous methods. Computational time for the alternate method is found to be 1.1308 seconds averaged over 100 runs. The Index-2 error is depicted with the static error in Figure (13).

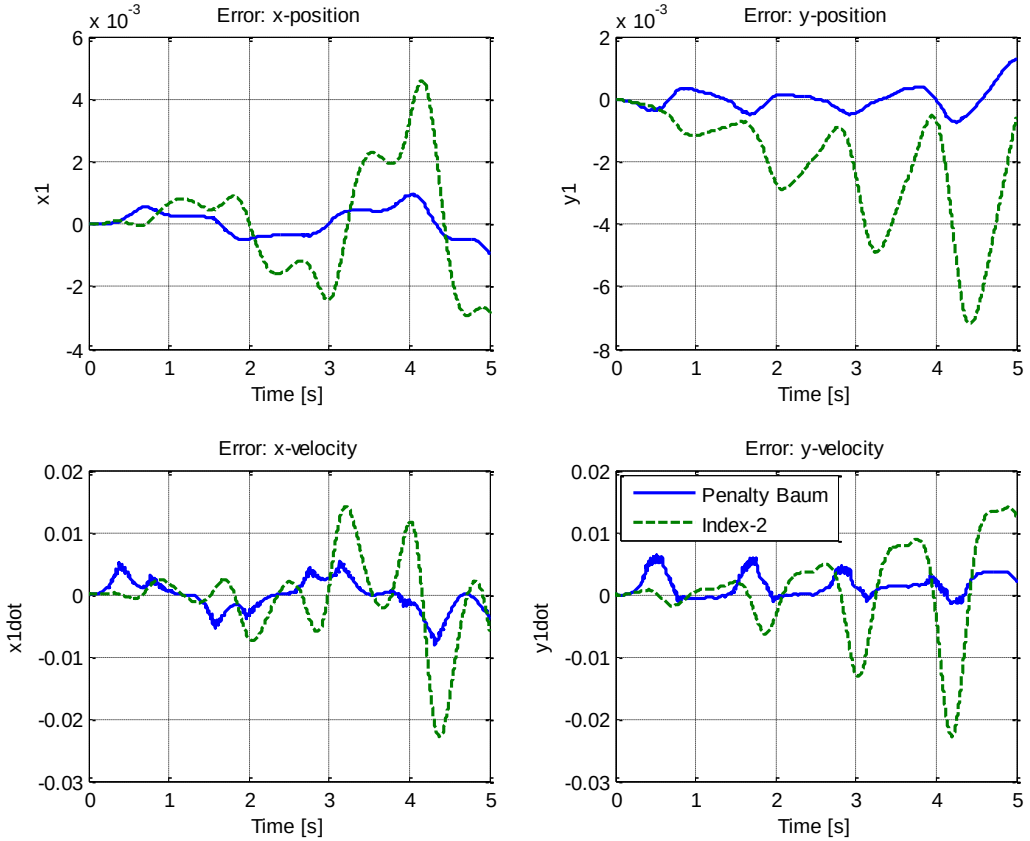

Figure 13: Index-2 Penalty Error

It is observed from all of the comparisons that the static penalty method applied to a dynamic problem is computationally feasible and accurate when compared with other proven methods. This holds true for both scenarios with and without the Baumgarte stabilization applied. The simple pendulum demonstrates this capability for a constrained single body system. The following examples illustrate the static penalty methods applicability with constrained multibody systems. In the following examples, only the projection method with Baumgarte stabilization will be utilized for comparison against the static penalty method with Baumgarte stabilization, since results for all methods are similar. Codes for the alternate

penalty and index-2 penalty can be easily manipulated to solve additional problems for interested readers.

2.4.2 SCOTCH MECHANISM

The first multibody problem chosen for this research is the Scotch mechanism illustrated in Figure (14). Gravity is acting uniformly over the system in the negative y direction. The problem is separated into two distinct bodies. An external force is applied to this system on body 2. The body-fixed reference frames for each body are depicted in Figures (15)-(16).

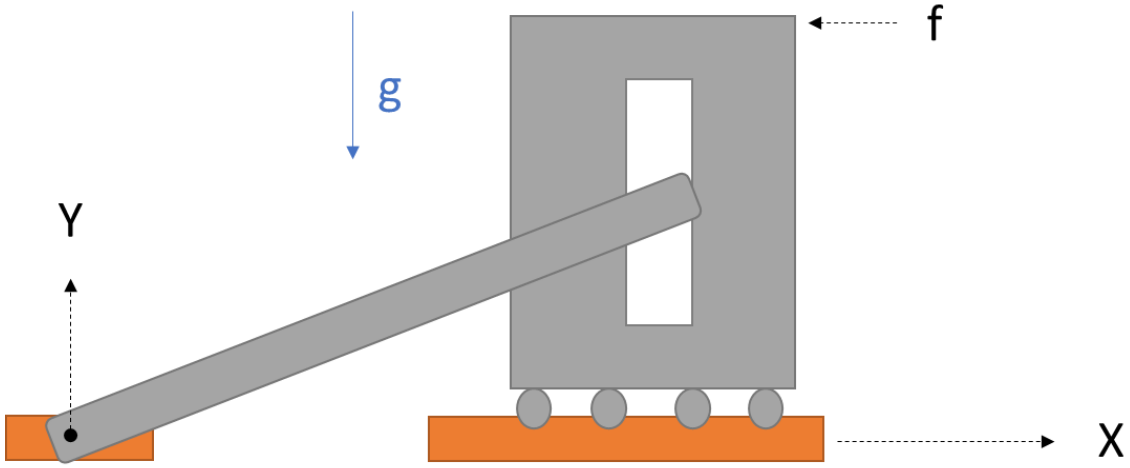

Figure 14: Scotch Mechanism

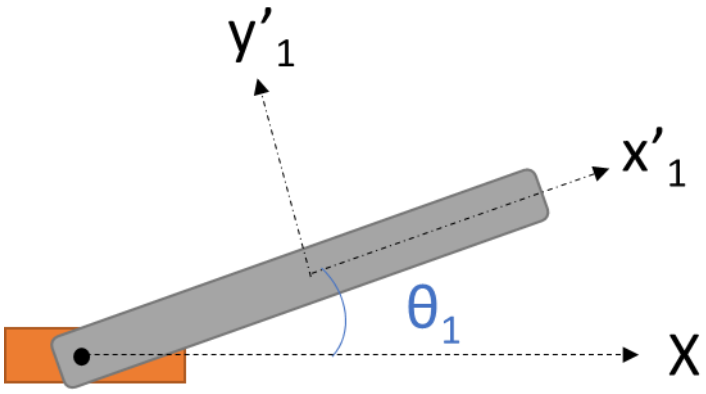

Figure 15: Scotch Mech. Body 1

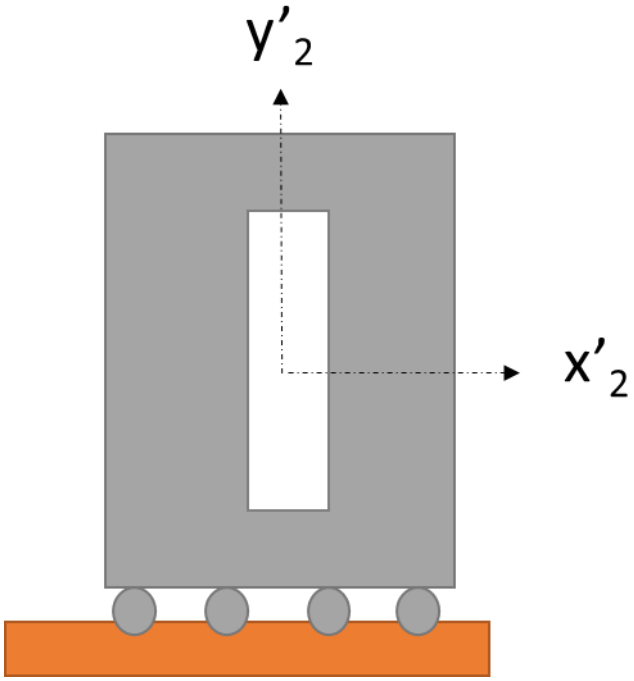

Figure 16: Scotch Mech. Body 2

The external force applied to body 2 will be defined as

$$f = -\sin(\frac{\pi}{4}t)$$

Use of equation (4) and (5) yield

$$\boldsymbol{M} = \begin{bmatrix} \frac{(4L_1^2 + b_1^2)}{12} m_1 & 0 \\ 0 & m_2 \end{bmatrix}$$

$$\boldsymbol{F} = \begin{bmatrix} -\frac{1}{2} m_1 g L_1 \cos\theta_1 \\ -f \end{bmatrix}$$

Constraints in Cartesian coordinates for the Scotch mechanism constrain body two to move only in the x-direction and body one to move within the slot on body 2. Holonomic constraints from Equation (19) are formulated as

$$\boldsymbol{C} = [x_2 - L_1 \cos(\theta_1)]$$

The Jacobian of the $\boldsymbol{C}$ matrix is then found using equation (20).

$$\boldsymbol{C}_{,\boldsymbol{q}} = [L_1 \sin(\theta_1) \quad 1]$$

Equation (22) is then used to calculate $\boldsymbol{\gamma}$:

$$\boldsymbol{\gamma} = [-L_1 \cos(\theta_1)\, \dot{\theta}_1^2]$$

These variables are then substituted into Equation (21) to formulate the DAE of the constrained Scotch mechanism, where

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{x}_2 \end{bmatrix}$$

Initial conditions for the Scotch mechanism are listed in Table 3. Initial conditions and variables are then substituted into the MATLAB® algorithm described in Section 2.2. The MATLAB® script

for this problem is depicted in Appendix A.3 and calls ODE23 functions seen in Appendix A.3.a and A.3.b. Solutions are compared with the example problem solution.

Table 3: Scotch Mech. Initial Conditions

g [in/s^2]	m_1 [lbf]	m_2 [lbf]	
386.4	0.5	5	
L_1 [in]	θ_1[rad]	x_2[in]	
25	1	13.5076	
b [in]	$\dot{\theta}_1$[rad/s]	$\dot{x}_2$[in/s]	
2	0	0	
t_{init} [s]	dt [s]	t_{final} [s]	Constraint Tolerance
0	0.0001	4	1^{-10}

Figures (17) illustrates Scotch mechanism motion as a result of the penalty method. For this case, a penalty factor (μ) of 10^7 is implemented on the constraints due to the maximum mass matrix component equaling 104. Figure (18) depicts Scotch mechanism motion using the projection method. The simulation is run for 4 seconds.

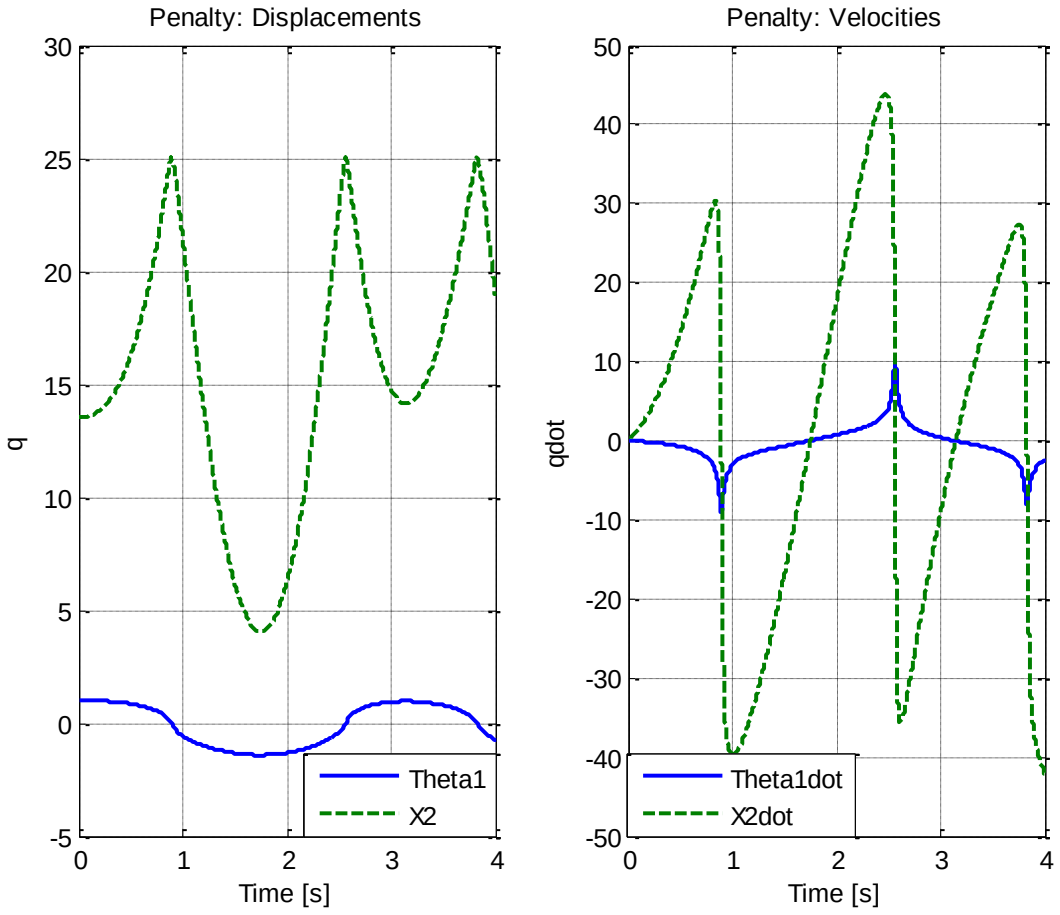

Figure 17: Scotch Penalty Motion

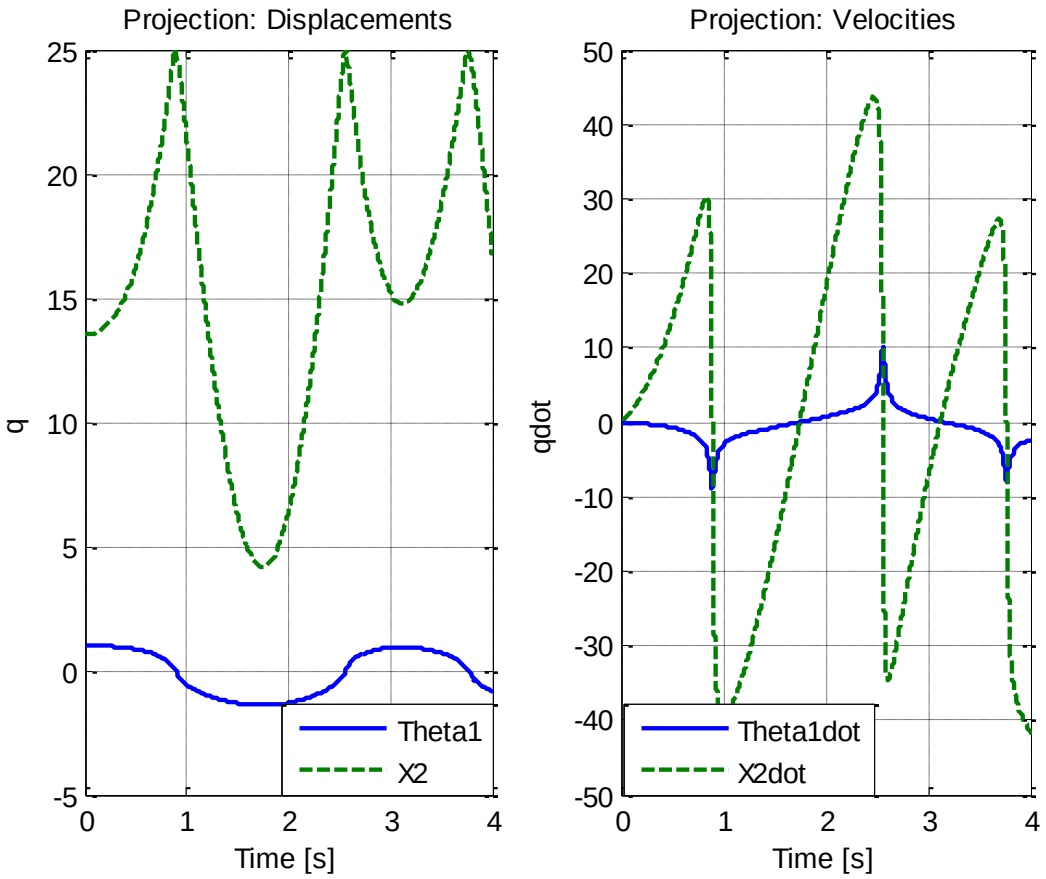

Figure 18: Scotch Projection Motion

The difference between the two methods is indistinguishable when comparing the above figures. Therefore, the difference between the two methods for each parameter is evaluated separately in Figure (19).

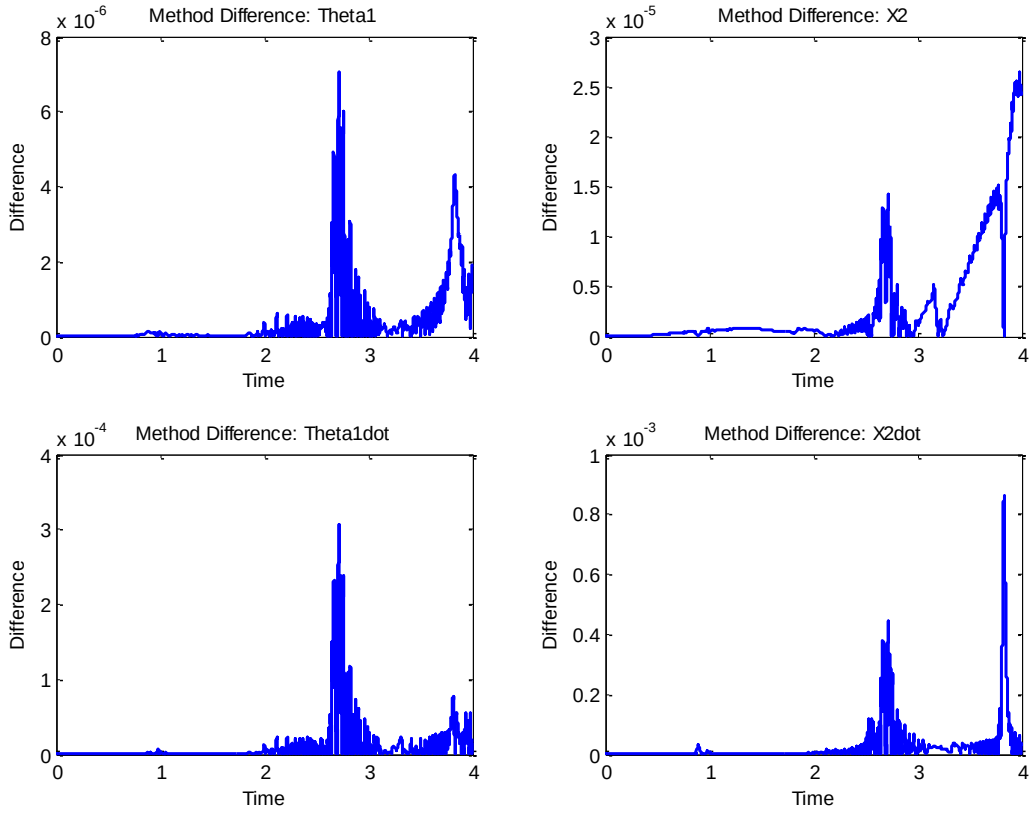

Figure 19: Scotch Penalty vs. Projection

Mean computational times over 100 runs were also found as 0.7528 seconds for the penalty method and 0.7214 for the Projection method. Since there is minimal deviation between the two methods in both solutions and computational time, the static penalty method is shown as a viable method for use with constrained multibody dynamics. To further assess the applicability of the penalty method for use with multibody dynamics, a more complex problem is analyzed.

2.4.3 SLIDER-CRANK MECHANISM

The slider-crank mechanism illustrated in Figure (20) provides a more complex system than that of the Scotch mechanism. Much like the Scotch mechanism, the slider-crank mechanism is commonly used in constrained multibody dynamics due to its variety of components and applicability towards formulations that can be used for other systems. Gravity is acting uniformly over the system in the negative y direction with no external torques or forces. The problem is separated into three distinct bodies and two constraints yielding one degree of freedom. Each symbol depicts its corresponding body with a subscript. The body-fixed reference frames for each body are depicted in Figures (21)-(23).

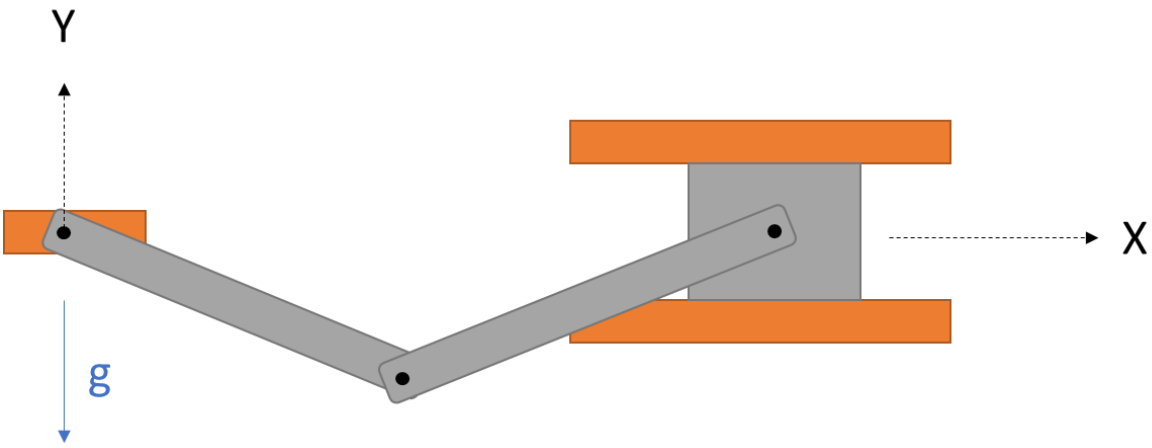

Figure 20: Slider-Crank Mechanism

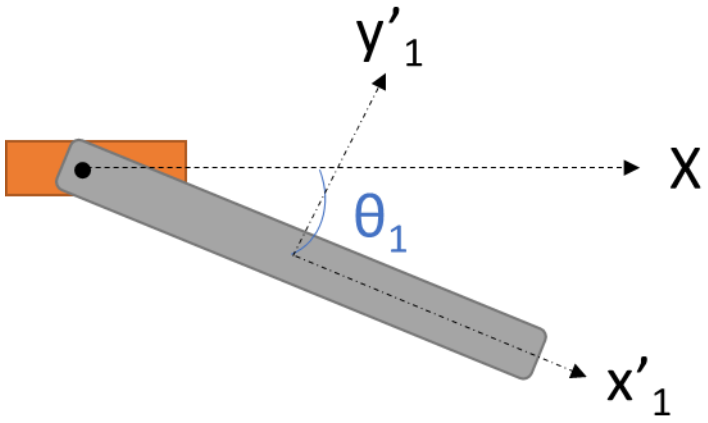

Figure 21: Slider-Crank Body 1

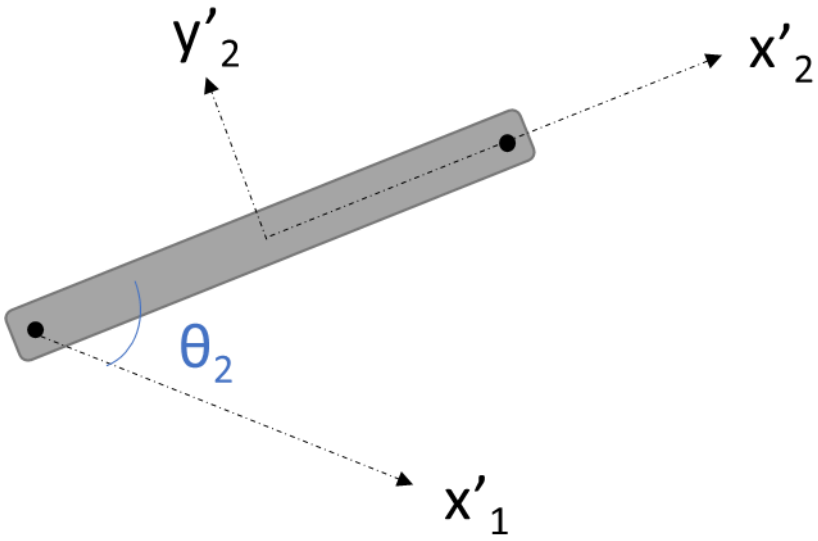

Figure 22: Slider-Crank Body 2

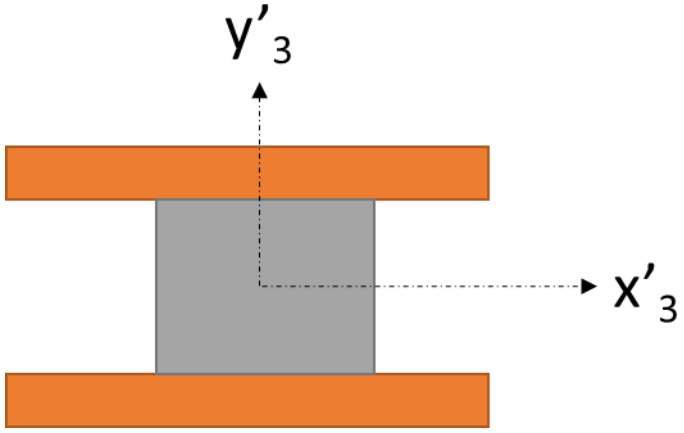

Figure 23: Slider-Crank Body 3

Use of equation (4) and (5) with the known inertia of rods at bodies 1 and 2 yields

$$\boldsymbol{M} = \begin{bmatrix} \frac{4}{12} m_1 L_1^2 & 0 & 0 \\ 0 & \frac{4}{12} m_2 L_2^2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}$$

$$\boldsymbol{F} = \begin{bmatrix} 2m_2 \sin(\theta_2)(\dot{\theta}_2^2 + 2\dot{\theta}_1\dot{\theta}_2) - m_2 g(2\sin(\theta_1) + \sin(\theta_1 + \theta_2)) - m_1 g \sin(\theta_1) \\ -2m_2 \sin(\theta_2)\,\dot{\theta}_1^2 - m_2 g \sin(\theta_1 + \theta_2) \\ 0 \end{bmatrix}$$

Constraints in Cartesian coordinates for the slider-crank mechanism restrict body three to move only in the x-direction. Holonomic constraints from Equation (19) are then formulated as

$$\boldsymbol{C} = \begin{bmatrix} L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) - x_3 \\ -L_1 \cos(\theta_1) - L_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$

The Jacobian of the ***C*** matrix is then found using equation (20).

$$\boldsymbol{C}_{,\boldsymbol{q}} = \begin{bmatrix} L_1\cos(\theta_1) + L_2\cos(\theta_1+\theta_2) & L_2\cos(\theta_1+\theta_2) & -1 \\ L_1\sin(\theta_1) + L_2\sin(\theta_1+\theta_2) & L_2\sin(\theta_1+\theta_2) & 0 \end{bmatrix}$$

Equation (22) is then used to calculate $\boldsymbol{\gamma}$:

$$\boldsymbol{\gamma} = -\begin{bmatrix} (L_1\sin(\theta_1) + L_2\sin(\theta_1+\theta_2))\dot{\theta}_1^2 + L_2\sin(\theta_1+\theta_2)\,(2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2) \\ -(L_1\cos(\theta_1) + L_2\cos(\theta_1+\theta_2))\dot{\theta}_1^2 - L_2\cos(\theta_1+\theta_2)\,(2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2) \end{bmatrix}$$

These variables are then substituted into Equation (21) to formulate the DAE of the constrained slider-crank mechanism, where

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{x}_3 \end{bmatrix}$$

Initial conditions for the slider-crank mechanism are listed in Table 4. Initial conditions were selected to match conditions from [8] for comparison to the literature. Initial conditions and variables are then substituted into the MATLAB® algorithm described in Section 2.2. The MATLAB® script for this problem is depicted in Appendix A.4 and calls ODE23 functions seen in Appendix A.4.a and A.4.b.

Table 4: Slider-Crank Initial Conditions

g [m/s^2]	m_1 [kg]	m_2 [kg]	m_3 [kg]
9.81	1	1	1
L_1 [m]	θ_1[rad]	θ_2[rad]	x_3 [m]
2	2.3562	-1.5708	2.8284
L_2 [m]	$\dot{\theta}_1$[rad/s]	$\dot{\theta}_2$[rad/s]	$\dot{x}_3$ [m/s]
2	0	0	0
t_{init} [s]	dt [s]	t_{final} [s]	Constraint Tolerance
0	0.0001	1.8	1^{-10}

Figures (24) illustrates slider-crank motion as a result of the penalty method. For this case, a penalty factor (μ) of 10^5 is implemented upon the constraints. Figure (25) depicts slider-crank motion using the projection method. The simulation is run for 1.8 seconds.

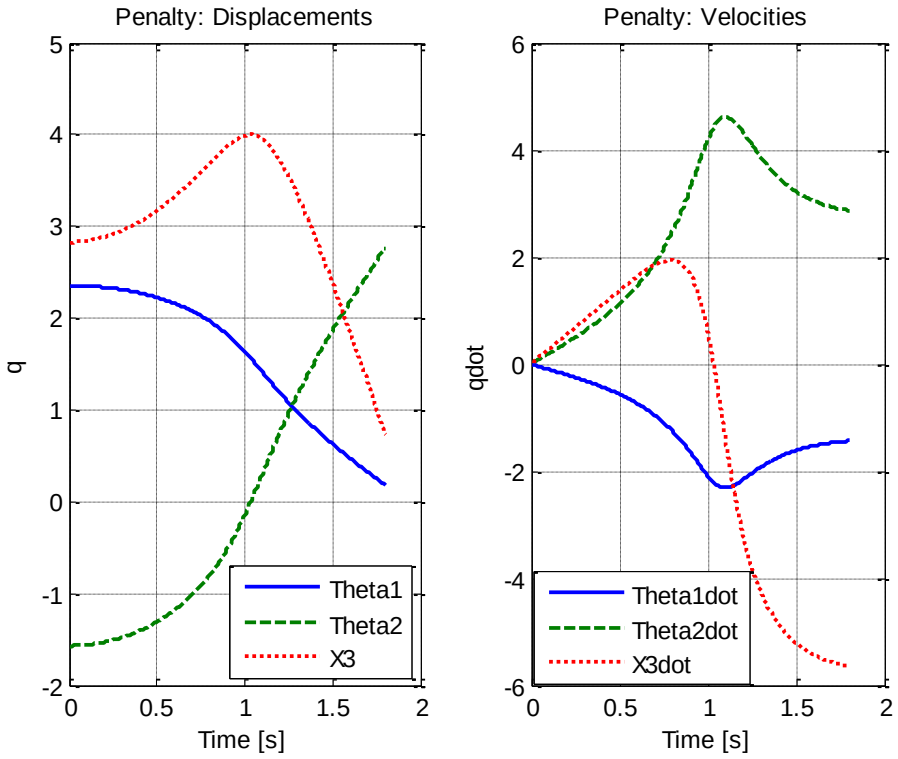

Figure 24: Slider-Crank Penalty Motion

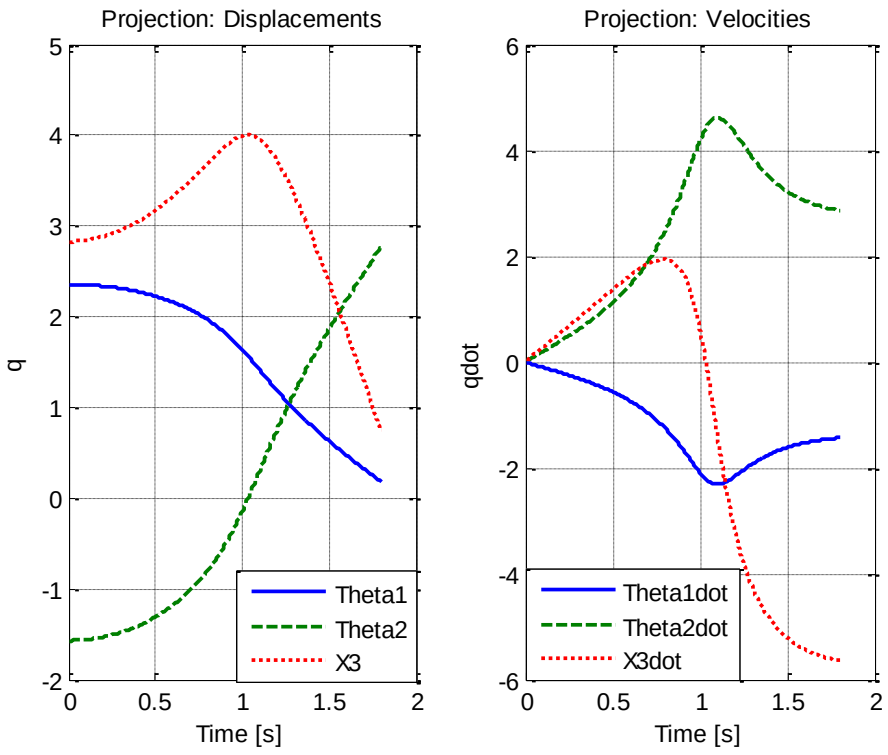

Figure 25: Slider-Crank Projection Motion

Computational times are again averaged over 100 computations for the two methods, yielding 0.3085 for the Projection method and 0.3141 for the penalty method. Solution difference between each method is illustrated in Figure (26).

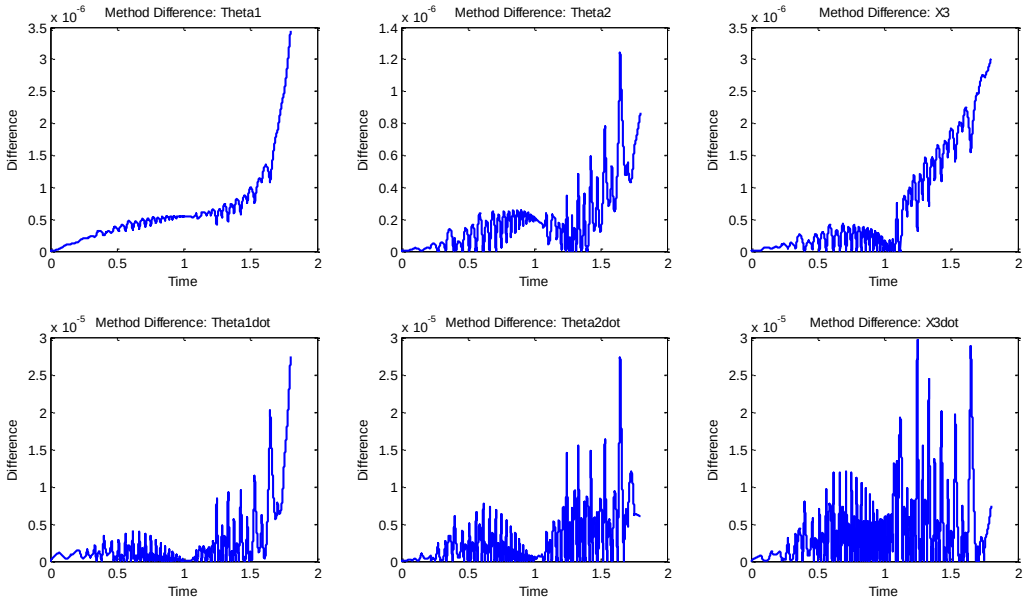

Figure 26: Slider-Crank Penalty vs. Projection

These differences are small comparatively. This demonstrates continued accuracy of the penalty method and computational efficiency that is nearly equivalent to the projection method.

CHAPTER 3. CONCLUSION AND FUTURE WORK

The objective of this study was to extend a static penalty method for use with constrained multibody dynamics and examine its ease of implementation within computer coding. The static penalty method was applied to example problems to verify its feasibility for use with constrained multibody dynamics. Example problems were selected with the intent to encompass a variety of multibody dynamic aspects. Chapter 2 examples illustrate the feasibility of extending the static penalty method presented in [13] for application with constrained multibody dynamic systems. Due to the complex and numerical nature of multibody dynamic systems, an algorithm was developed which solves a system using the static penalty method. Penalty solutions were compared with direct numerical solutions, projection solutions [12], an alternate penalty solution [16], an Index-2 penalty solution, and literature. Favorable results were obtained for the penalty method based on solution accuracy and computational time. A Baumgarte stabilization [17] was applied to the static penalty method as an alternate enforcement of hidden constraints. It was found that the Baumgarte stabilization does not negatively impact accuracy or computational time. Using the stabilization results in significantly less code required for solution algorithms. Implementation of the penalty method within MATLAB® is relatively simple, especially with Baumgarte stabilization. It is also shown that the ideal penalty factor is 5 orders of magnitude greater than the largest value observed in a formulated mass matrix. The applicability of the static penalty method was demonstrated for systems that can easily be extrapolated to solve many other dynamic DAEs. Overall, the objective of this thesis was met.

Example problems were selected with the intent to encompass a variety of multibody dynamic aspects. In making these selections, the presented penalty method can be manipulated to solve any DAE multibody system with holonomic constraints. A projection method was used to compare solutions. The projection method was chosen for comparison due to its similarity with implementation in coding and its proven solution accuracy [12].

The first example illustrates the feasibility of using the penalty method for constrained dynamic analysis. In comparing the static penalty, projection [12], alternate penalty [16], and Index-2 penalty methods with a direct numerical solution, similar accuracies and computational times are observed. Additionally, a Baumgarte constraint is applied to the static penalty and projection methods as an alternate way of enforcing the system's hidden constraints. The two methods are then compared with and without this stabilization, resulting in nearly indistinguishable results. MATLAB® code is significantly reduced when using the Baumgarte stabilization, leading to its use for subsequent example problems. These results prove that it is feasible to use the presented static penalty method for constrained dynamics.

In order to test the static penalty method with a constrained multibody system, the Scotch mechanism was chosen due to its significance in multibody dynamics and component variety. The static penalty method for this problem is compared directly with the projection method, both of which have the hidden constraints enforced using Baumgarte stabilization. It is observed that the static penalty solution is well within an acceptable tolerance of the projection solution and requires similar computational time. This analysis shows that the static penalty method is feasible for use with constrained multibody dynamics.

The final problem depicts solutions for a slider-crank mechanism. The static penalty method for this problem is compared directly with the projection method and solutions presented by [8]. Again, the Baumgarte stabilization is utilized by both methods to enforce hidden constraints. It is observed that the static penalty solution is well within an acceptable tolerance of the projection solution and requires similar computational time. This analysis shows the robustness of the static penalty method, even as problem complexity increases.

In comparing the static penalty and projection method, it is seen from the example problems that computational times and accuracies are similar. The derivation for the projection method is a much more rigorous process, and its implementation within MATLAB® code more difficult than that of the penalty method. The ease with which the penalty method can be implemented and understood in MATLAB® code is beneficial, especially as increasingly complex systems are observed.

Although the static penalty method proves useful for constrained multibody systems, additional research is required to determine its applicability and overall performance with increasingly complex systems. Increasingly complex systems may contain additional bodies, non-holonomic constraints, Euler parameters, etc. Computationally, the performance of any method is going to diminish as the number of bodies and constraints increase. This is simply due to the number of calculations that must be performed. Based on the limited number of examples presented in this research, there is evidence that the deviation between penalty and projection method does not change. This insinuates that the penalty method could be used in lieu of the projection method. Lastly, the DAE solutions for this research encompass constrained multibody dynamics. DAE formulations are not limited to this field of study.

Solutions of DAEs for other disciplines (i.e. fluid dynamics, heat transfer) has yet to be assessed using the presented static penalty method.

REFERENCES

[1] Yang, Yinping and Peter Betsch. "On the Choice of Coordinates forComputational Multibody Dynamics." Proceedings in Applied Mathematics & Mechanics (2011): 75-76.

[2] Haug, Edward J. Intermediate Dynamics. Englewood Cliffs, NJ: Prentice-Hall Inc., 1992.

[3] Greenber, Michael D. Advanced Engineering Mathematics. Upper Saddle River, NJ: Pearson Prentice Hall, 1998.

[4] Weisstein, Eric W. "Ordinary Differential Equations." n.d. Wolfram. <http://mathworld.wolfram.com/OrdinaryDifferentialEquation.html>.

[5] Moler, Cleve. "Stiff Differential Equations." 2003. MathWorks. <https://www.mathworks.com/company/newsletters/articles/stiff-differential-equations.html>.

[6] Petzold, Linda. "Differential/Algebraic Equations Are Not ODE's." SIAM Journal on Scientific Computing, 1982: 367-384.

[7] Goldstein, Herbert, Charles Poole and John Safko. Classiscal Mechanics. Addison-Wesley, 2000.

[8] Haug, Edward J. "Computational Methods in Mechanical System Dynamics." June 2019. Research Gate. <www.researchgate.net>.

[9] Negrut, Dan. On the Implicit Integration of Differential-Algebraic Equations of Multibody Dynamics. PhD Thesis. Iowa City, Iowa: University of Iowa, 1998.

[10] Hibbeler, R.C. Engineering Mechanics: Dynamics. Upper Saddle River, NJ: Pearson Prentice Hall, 2013.

[11] Okasha, M. and B. Newman. "Switching Principles to Circumvent Euler Angle Singularity." AIAA/AAS Astrodynamics Specialist Conference. Toronto, Canada, 2010.

[12] Heaney, Patrick S. and Gene Hou. "Projection Method with Minimal Correction Procedure for Numerical Simulation of Constrained Dynamics." ASME 2017 Dynamic Systems and Control Conference. Tysons Corner, VA, 2017: 1-10.

[13] Chandrupatla, Tirupathi R. and Ashok D. Belegundu. "Penalty Approach." Introduction. Upper Saddle River, NJ: Pearson Prentice Hall, 2012: 76-85.

[14] Shampine, L.F. and M.W. Reichelt. "The MATLAB ODE Suite." SIAM Journal on Scientific Computing (1997): 1-22.

[15] Bogacki, P. and Shampine, L.F. "A 3(2) Pair of Runge-Kutta Formulas" Appl. Math. Lett., 1989: Vol. 2, 321-325.

[16] Goicolea, J. M. and Garcia Orden J. C. , “Dynamic Analysis of Rigid and Deformable Multibody Systems with Penalty methods and Energy-Momentum Schemes”, Computer Methods in Applied Mechanics and Engineering , 2000: Vol. 188, 789-804.

[17] Hajzman, M. and Polach, P., “Application of stabilization techniques in the dynamic analysis of multibody systems”, Applied and Computational Mechanics, 2007: Vol. 1, 479-488.

APPENDIX A. COMPUTER CODES

A.1. Static Multipoint Constraint

```
close all; clear all;
set(0,'defaultlinelinewidth',2);

%% Static Penalty Method

E1 = 200*10^3; E2 = 70*10^3; % Modulus of elasticity [N/mm^2]
A1 = 1200; A2 = 900; % Cross sectional area [mm^2]
L1 = 4500; L2 = 3000; % Length [mm]
P = 30*10^3; % Applied load [N]

F = [0;0;0;0;P]; % External forces on the bar [N]

k1 = E1*A1/L1*[1 -1;-1 1]; % Body 1 stiffness matrix
k2 = E2*A2/L2*[1 -1;-1 1]; % Body 2 stiffness matrix

K = [k1(2,2) 0 k1(2,1) 0 0;
    0 k2(2,2) 0 k2(2,1) 0;
    k1(1,2) 0 k1(1,1) 0 0;
    0 k2(1,2) 0 k2(1,1) 0;
    0 0 0 0 0]; % Global stiffness matrix

MK = max(K);

mu = MK(1)*10^4; % Constraint

C = [1 0 0 0 -2/6;0 1 0 0 -5/6]; % Constraints keeping bar straight
CC = [0 0 0 0 0;0 0 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 0]; % Attachment
Constraints
PF = mu*((C'*C)+CC);

K_pen = K+PF;

Q = K_pen\F; % Calculated Displacements [mm]
```

A.2. Simple Pendulum

```
close all; clear all;
set(0,'defaultlinelinewidth',2);
% Simple pendulum motion

%% Parameters
% Time span parameters [seconds]
tinit = 0;
tstep = 0.0001;
tfinal = 5;

% Model
global g m1 tp mu alpha beta
```

```
g = 9.81; % [m/s^2] gravitational constant
m1 =1; % [kg] mass of the pendulum
tp = tinit:tstep:tfinal; % [sec] Time span for evaluation
mu = 10^5; % Finite Element Penalty Factor
alpha = 20; % Baumgarte Weighting Coefficient
beta = 20; % Baumgarte Weighting Coefficient

% Initial Conditions
x1initial = 1; y1initial = 0; zinitial = 0; % [m] Length is 1
x1dotinitial = 0; y1dotinitial = 0; zdotinitial = 0; % [m/s]

q0(1) = x1initial;
q0(2) = y1initial;
q0(3) = zinitial;
q0(4) = x1dotinitial;
q0(5) = y1dotinitial;
q0(6) = zdotinitial;

%% Projection Method
tic
disp('Solution Method: Projection');
[t,qp]=ode23(@Pendulum_ProjectionFunc,  tp, q0);
Projtimer = toc

% Process and Plot Solution for Projection method

figure;
subplot(2,2,1); plot(t,qp(:,1)); xlabel('Time [s]'); ylabel('x1'); grid on;
xlim([0 5]); ylim([-1 1]);
title('Projection: x-position');
subplot(2,2,2); plot(t,qp(:,2)); xlabel('Time [s]'); ylabel('y1'); grid on;
xlim([0 5]); ylim([0 1]);
title('Projection: y-position');
subplot(2,2,3); plot(t,qp(:,4)); xlabel('Time [s]'); ylabel('x1dot'); grid
on;
xlim([0 5]); ylim([-5 5]);
title('Projection: x-velocity');
subplot(2,2,4); plot(t,qp(:,5)); xlabel('Time [s]'); ylabel('y1dot'); grid
on;
xlim([0 5]); ylim([-4 4]);
title('Projection: y-velocity');

%% Projection Method Baumgarte Stabilization
tic
disp('Solution Method: Projection Baumgarte');
[t,qpb]=ode23(@Pendulum_ProjectionBaum,  tp, q0);
ProjBaumTimer = toc

% Process and Plot Solution for Projection method

figure;
subplot(2,2,1); plot(t,qpb(:,1)); xlabel('Time [s]'); ylabel('x1'); grid on;
xlim([0 5]); ylim([-1 1]);
title('Projection Baum: x-position');
subplot(2,2,2); plot(t,qpb(:,2)); xlabel('Time [s]'); ylabel('y1'); grid on;
```

```
xlim([0 5]); ylim([0 1]);
title('Projection Baum: y-position');
subplot(2,2,3); plot(t,qpb(:,4)); xlabel('Time [s]'); ylabel('x1dot'); grid
on;
xlim([0 5]); ylim([-5 5]);
title('Projection Baum: x-velocity');
subplot(2,2,4); plot(t,qpb(:,5)); xlabel('Time [s]'); ylabel('y1dot'); grid
on;
xlim([0 5]); ylim([-4 4]);
title('Projection Baum: y-velocity');

%% Finite Element Penalty Method
tic
disp('Solution Method: Static Penalty');
[t,q]=ode23(@Pendulum_PenaltyFunc,  tp, q0);
PenaltyFiniteTimer = toc

% Process and Plot Solution for Finite Element Penalty Method

figure;
subplot(2,2,1); plot(t,q(:,1)); xlabel('Time [s]'); ylabel('x1'); grid on;
xlim([0 5]); ylim([-1 1]);
title('Static Penalty: x-position');
subplot(2,2,2); plot(t,q(:,2)); xlabel('Time [s]'); ylabel('y1'); grid on;
xlim([0 5]); ylim([0 1]);
title('Static Penalty: y-position');
subplot(2,2,3); plot(t,q(:,4)); xlabel('Time [s]'); ylabel('x1dot'); grid on;
xlim([0 5]); ylim([-5 5]);
title('Static Penalty: x-velocity');
subplot(2,2,4); plot(t,q(:,5)); xlabel('Time [s]'); ylabel('y1dot'); grid on;
xlim([0 5]); ylim([-4 4]);
title('Static Penalty: y-velocity');

%% Finite Element Penalty Method with Baumgarte Stabilization
tic
disp('Solution Method: Static Penalty Baumgarte');
[t,qb]=ode23(@Pendulum_FinitePenaltyBaum,  tp, q0);
BaumFiniteTimer = toc

% Process and Plot Solution for Finite Element Penalty Method Baumgarte

figure;
subplot(2,2,1); plot(t,qb(:,1)); xlabel('Time [s]'); ylabel('x1'); grid on;
xlim([0 5]); ylim([-1 1]);
title('Static Baumgarte: x-position');
subplot(2,2,2); plot(t,qb(:,2)); xlabel('Time [s]'); ylabel('y1'); grid on;
xlim([0 5]); ylim([0 1]);
title('Static Baumgarte: y-position');
subplot(2,2,3); plot(t,qb(:,4)); xlabel('Time [s]'); ylabel('x1dot'); grid
on;
xlim([0 5]); ylim([-5 5]);
title('Static Baumgarte: x-velocity');
subplot(2,2,4); plot(t,qb(:,5)); xlabel('Time [s]'); ylabel('y1dot'); grid
on;
xlim([0 5]); ylim([-4 4]);
title('Static Baumgarte: y-velocity');
```

```
%% Goicolea and Garcia-Orden Penalty Method
tic
disp('Solution Method: Goicolea Penalty');
[t,qg]=ode23(@Pendulum_GOPenaltyFunc,  tp, q0);
PenaltyGoicoleaTimer = toc

% Process and Plot Solution for Finite Element Penalty Method

figure;
subplot(2,2,1); plot(t,qg(:,1)); xlabel('Time [s]'); ylabel('x1'); grid on;
xlim([0 5]); ylim([-1 1]);
title('Goicolea Penalty: x-position');
subplot(2,2,2); plot(t,qg(:,2)); xlabel('Time [s]'); ylabel('y1'); grid on;
xlim([0 5]); ylim([0 1]);
title('Goicolea Penalty: y-position');
subplot(2,2,3); plot(t,qg(:,4)); xlabel('Time [s]'); ylabel('x1dot'); grid
on;
xlim([0 5]); ylim([-5 5]);
title('Goicolea Penalty: x-velocity');
subplot(2,2,4); plot(t,qg(:,5)); xlabel('Time [s]'); ylabel('y1dot'); grid
on;
xlim([0 5]); ylim([-4 4]);
title('Goicolea Penalty: y-velocity');

%% Goicolea and Garcia-Orden Penalty Method
tic
disp('Solution Method: Index-2 Penalty');
[t,qi2]=ode23(@Pendulum_Index2Penalty,  tp, q0);
PenaltyIndex2Timer = toc

% Process and Plot Solution for Finite Element Penalty Method

figure;
subplot(2,2,1); plot(t,qi2(:,1)); xlabel('Time [s]'); ylabel('x1'); grid on;
xlim([0 5]); ylim([-1 1]);
title('Index 2 Penalty: x-position');
subplot(2,2,2); plot(t,qi2(:,2)); xlabel('Time [s]'); ylabel('y1'); grid on;
xlim([0 5]); ylim([0 1]);
title('Index 2 Penalty: y-position');
subplot(2,2,3); plot(t,qi2(:,4)); xlabel('Time [s]'); ylabel('x1dot'); grid
on;
xlim([0 5]); ylim([-5 5]);
title('Index 2 Penalty: x-velocity');
subplot(2,2,4); plot(t,qi2(:,5)); xlabel('Time [s]'); ylabel('y1dot'); grid
on;
xlim([0 5]); ylim([-4 4]);
title('Index 2 Penalty: y-velocity');

%% Direct Integration

% Initial Condition
theta0 = pi/2;
omega0 = 0;
L = 1;
```

```
disp('Solution Method: Direct');

qe0(1) = theta0;
qe0(2) = omega0;

tic
disp('Solution Method: Exact Penalty');
[t,qe]=ode23(@Pendulum_Direct,  tp, qe0);
DirectTimer = toc

x = L.*sin(qe(:,1));
y = L.*cos(qe(:,1));
xdot = L.*qe(:,2).*cos(qe(:,1));
ydot = -L.*qe(:,2).*sin(qe(:,1));

figure;
subplot(2,2,1); plot(tp,x); xlabel('Time [s]'); ylabel('x1'); grid on;
xlim([0 5]); ylim([-1 1]);
title('Exact: x-position');
subplot(2,2,2); plot(tp,y); xlabel('Time [s]'); ylabel('y1'); grid on;
xlim([0 5]); ylim([0 1]);
title('Exact: y-position');
subplot(2,2,3); plot(tp,xdot); xlabel('Time [s]'); ylabel('x1dot'); grid on;
xlim([0 5]); ylim([-5 5]);
title('Exact: x-velocity');
subplot(2,2,4); plot(tp,ydot); xlabel('Time [s]'); ylabel('y1dot'); grid on;
xlim([0 5]); ylim([-4 4]);
title('Exact: y-velocity');

%% Calculate the error compared with the direct solution

Prox_error = (qp(:,1)-x);
Proy_error = (qp(:,2)-y);
Proxdot_error = (qp(:,4)-xdot);
Proydot_error = (qp(:,5)-ydot);

ProBaumx_error = (qpb(:,1)-x);
ProBaumy_error = (qpb(:,2)-y);
ProBaumxdot_error = (qpb(:,4)-xdot);
ProBaumydot_error = (qpb(:,5)-ydot);

Penx_error = (q(:,1)-x);
Peny_error = (q(:,2)-y);
Penxdot_error = (q(:,4)-xdot);
Penydot_error = (q(:,5)-ydot);

PenBaumx_error = (qb(:,1)-x);
PenBaumy_error = (qb(:,2)-y);
PenBaumxdot_error = (qb(:,4)-xdot);
PenBaumydot_error = (qb(:,5)-ydot);

PenGOx_error = (qg(:,1)-x);
PenGOy_error = (qg(:,2)-y);
PenGOxdot_error = (qg(:,4)-xdot);
PenGOydot_error = (qg(:,5)-ydot);
```

```
PenI2x_error = (qi2(:,1)-x);
PenI2y_error = (qi2(:,2)-y);
PenI2xdot_error = (qi2(:,4)-xdot);
PenI2ydot_error = (qi2(:,5)-ydot);

% Plot Errors with the direct solution

figure;
subplot(2,2,1); plot(t,Penx_error,t,Prox_error,'--',t,ProBaumx_error,'--
',t,PenBaumx_error,':',t,PenGOx_error,'-.'); xlabel('Time [s]');
ylabel('x1'); grid on;
title('Error: x-position');
subplot(2,2,2); plot(t,Peny_error,t,Proy_error,'--',t,ProBaumy_error,'--
',t,PenBaumy_error,':',t,PenGOy_error,'-.'); xlabel('Time [s]');
ylabel('y1'); grid on;
title('Error: y-position');
subplot(2,2,3); plot(t,Penxdot_error,t,Proxdot_error,'--
',t,ProBaumxdot_error,'--',t,PenBaumxdot_error,':',t,PenGOxdot_error,'-.');
xlabel('Time [s]'); ylabel('x1dot'); grid on;
title('Error: x-velocity');
subplot(2,2,4); plot(t,Penydot_error,t,Proydot_error,'--
',t,ProBaumydot_error,'--',t,PenBaumydot_error,':',t,PenGOydot_error,'-.');
xlabel('Time [s]'); ylabel('y1dot'); grid on;
title('Error: y-velocity');
legend('Finite Penalty','Projection','Projection Baumgarte','Penalty
Baumgarte','Goicolea Penalty');

figure;
subplot(2,2,1); plot(t,PenBaumx_error,':',t,ProBaumx_error); xlabel('Time
[s]'); ylabel('x1'); grid on;
title('Error: x-position');
subplot(2,2,2); plot(t,PenBaumy_error,':',t,ProBaumy_error); xlabel('Time
[s]'); ylabel('y1'); grid on;
title('Error: y-position');
subplot(2,2,3); plot(t,PenBaumxdot_error,':',t,ProBaumxdot_error);
xlabel('Time [s]'); ylabel('x1dot'); grid on;
title('Error: x-velocity');
subplot(2,2,4); plot(t,PenBaumydot_error,':',t,ProBaumydot_error);
xlabel('Time [s]'); ylabel('y1dot'); grid on;
title('Error: y-velocity');
legend('Penalty Baumgarte','Projection Baumgarte');

%% Difference between methods
% Finite Penalty with and without Baumgarte Stabilization
Baumx_diff = abs(q(:,1)-qb(:,1));
Baumy_diff = abs(q(:,2)-qb(:,2));
Baumxdot_diff = abs(q(:,4)-qb(:,4));
Baumydot_diff = abs(q(:,5)-qb(:,5));

figure;
subplot(2,2,1); plot(tp,Baumx_diff); xlabel('Time [s]'); ylabel('x1'); grid
on;
title('Baumgarte Penalty x-position Difference');
subplot(2,2,2); plot(tp,Baumy_diff); xlabel('Time [s]'); ylabel('y1'); grid
on;
```

```
title('Baumgarte Penalty y-position Difference');
subplot(2,2,3); plot(tp,Baumxdot_diff); xlabel('Time [s]'); ylabel('x1dot');
grid on;
title('Baumgarte Penalty x-velocity Difference');
subplot(2,2,4); plot(tp,Baumydot_diff); xlabel('Time [s]'); ylabel('y1dot');
grid on;
title('Baumgarte Penalty y-velocity Difference');

% Projection with and without Baumgarte Stabilization
Projx_diff = abs(qpb(:,1)-qp(:,1));
Projy_diff = abs(qpb(:,2)-qp(:,2));
Projxdot_diff = abs(qpb(:,4)-qp(:,4));
Projydot_diff = abs(qpb(:,5)-qp(:,5));

figure;
subplot(2,2,1); plot(tp,Projx_diff); xlabel('Time [s]'); ylabel('x1'); grid
on;
title('Baumgarte Projection x-position Difference');
subplot(2,2,2); plot(tp,Projy_diff); xlabel('Time [s]'); ylabel('y1'); grid
on;
title('Baumgarte Projection y-position Difference');
subplot(2,2,3); plot(tp,Projxdot_diff); xlabel('Time [s]'); ylabel('x1dot');
grid on;
title('Baumgarte Projection x-velocity Difference');
subplot(2,2,4); plot(tp,Projydot_diff); xlabel('Time [s]'); ylabel('y1dot');
grid on;
title('Baumgarte Projection y-velocity Difference');

% Finite Penalty and Projection with Baumgarte Stabilization
Methodx_diff = abs(qpb(:,1)-qb(:,1));
Methody_diff = abs(qpb(:,2)-qb(:,2));
Methodxdot_diff = abs(qpb(:,4)-qb(:,4));
Methodydot_diff = abs(qpb(:,5)-qb(:,5));

figure;
subplot(2,2,1); plot(tp,Methodx_diff); xlabel('Time [s]'); ylabel('x1'); grid
on;
title('BaumPen-BaumProj x-position Difference');
subplot(2,2,2); plot(tp,Methody_diff); xlabel('Time [s]'); ylabel('y1'); grid
on;
title('BaumPen-BaumProj y-position Difference');
subplot(2,2,3); plot(tp,Methodxdot_diff); xlabel('Time [s]');
ylabel('x1dot'); grid on;
title('BaumPen-BaumProj x-velocity Difference');
subplot(2,2,4); plot(tp,Methodydot_diff); xlabel('Time [s]');
ylabel('y1dot'); grid on;
title('BaumPen-BaumProj y-velocity Difference');

% Finite Penalty with Baumgarte and Goicolea Penalty
Methodx_diff = abs(qb(:,1)-qg(:,1));
Methody_diff = abs(qb(:,2)-qg(:,2));
Methodxdot_diff = abs(qb(:,4)-qg(:,4));
Methodydot_diff = abs(qb(:,5)-qg(:,5));

figure;
```

```
subplot(2,2,1); plot(tp,Methodx_diff); xlabel('Time [s]'); ylabel('x1'); grid
on;
title('Finite and GO x-position Difference');
subplot(2,2,2); plot(tp,Methody_diff); xlabel('Time [s]'); ylabel('y1'); grid
on;
title('Finite and GO y-position Difference');
subplot(2,2,3); plot(tp,Methodxdot_diff); xlabel('Time [s]');
ylabel('x1dot'); grid on;
title('Finite and GO x-velocity Difference');
subplot(2,2,4); plot(tp,Methodydot_diff); xlabel('Time [s]');
ylabel('y1dot'); grid on;
title('Finite and GO y-velocity Difference');
```

A.2.a. Simple Pendulum Penalty Function

```
function dqdt = Pendulum_PenaltyFunc(t,q) % Single Pendulum
% Static Penalty Method with corrections for hidden constraints
    constraintTolerance = 1e-10; % Tolerance for displacement and velocity
constraints
    global g m1 mu
    % Read current state
    x1 = q(1);
    y1 = q(2);
    z = q(3);
    x1dot = q(4);
    y1dot = q(5);
    zdot = q(6);

    C = [x1^2+y1^2-1;z];
    Cq = [2*x1 2*y1 0;0 0 1];

    %% Correct displacement constraint
    while abs(C) > constraintTolerance

        Q_delta = -Cq'*inv(Cq*Cq')*C;

        for i = 1:3
            q(i) = q(i) + Q_delta(i);
        end

        % Update displacement state
        x1=q(1); y1 = q(2); z = q(3);

    C = [x1^2+y1^2-1;0];
    Cq = [2*x1 2*y1 0;0 0 1];
    end

    C_dot = Cq*q(4:6);

   %% Correct velocity constraint
    while abs(C_dot) > constraintTolerance
```

```
        Q_dot_delta = -Cq'*inv(Cq*Cq')*C_dot;

        for i = 1:3
            q(i+3) = q(i+3) + Q_dot_delta(i);
        end

        % Update velocity state
        x1dot=q(4); y1dot = q(5); zdot = q(6);

        % Update velocity constraint
        C_dot = Cq*q(4:6);
    end

    %% Update acceleration
    M = [m1 0 0;0 m1 0;0 0 m1];

    gamma = [-2*x1dot^2-2*y1dot^2;0];

    f = [0;m1*g;0];

    qddot = inv(M+mu*Cq'*Cq)*(f+mu*Cq'*gamma);

    lamda = inv(Cq*inv(M)*Cq')*(Cq*inv(M)*f-gamma);

    dqdt = [q(4:6);qddot];
end
```

A.2.b. Simple Pendulum Penalty Function, Baumgarte Stabilization

```
function dqdt = Pendulum_FinitePenaltyBaum(t,qb) % Single Pendulum
% Static Penalty Method with Baumgarte Stabilization
    global g m1 mu alpha beta
    % Read current state
    x1 = q(1);
    y1 = q(2);
    z = q(3);
    x1dot = q(4);
    y1dot = q(5);
    zdot = q(6);

    C = [x1^2+y1^2-1;z];
    Cq = [2*x1 2*y1 0;0 0 1];
    C_dot = Cq*q(4:6);

    % Baumgarte Stabilization
    B = 2*alpha*C_dot + beta^2*C;


    %% Update acceleration
    M = [m1 0 0;0 m1 0;0 0 m1];

    gamma = [-2*x1dot^2-2*y1dot^2;0]+B;
```

```
    f = [0;m1*g;0];

    qddot = inv(M+mu*Cq'*Cq)*(f+mu*Cq'*gamma);

    lamda = inv(Cq*inv(M)*Cq')*(Cq*inv(M)*f-gamma);

    dqdt = [q(4:6);qddot];
end
```

A.2.c. Simple Pendulum Projection Function

```
function dqdt = Pendulum_ProjectionFunc(t,qp) % Single Pendulum
% Projection Method with corrections for hidden constraints
    constraintTolerance = 1e-10; % Tolerance for displacement and velocity
constraints
    global g m1
    % Read current state
    x1 = qp(1);
    y1 = qp(2);
    z = qp(3);
    x1dot = qp(4);
    y1dot = qp(5);
    zdot = qp(6);

    C = [x1^2+y1^2-1;z];
    Cq = [2*x1 2*y1 0;0 0 1];

    %% Correct displacement constraint
    while abs(C) > constraintTolerance

        Q_delta = -Cq'*inv(Cq*Cq')*C;

        for i = 1:3
            qp(i) = qp(i) + Q_delta(i);
        end

        % Update displacement state
        x1=qp(1); y1 = qp(2); z = qp(3);

    C = [x1^2+y1^2-1;0];
    Cq = [2*x1 2*y1 0;0 0 1];
    end

    C_dot = Cq*qp(4:6);

   %% Correct velocity constraint
    while abs(C_dot) > constraintTolerance

        Q_dot_delta = -Cq'*inv(Cq*Cq')*C_dot;

        for i = 1:3
            qp(i+3) = qp(i+3) + Q_dot_delta(i);
        end
```

```
        % Update velocity state
        x1dot=qp(4); y1dot = qp(5); zdot = qp(6);

        % Update velocity constraint
        C_dot = Cq*qp(4:6);
    end

    %% Update acceleration
    invM = inv([m1 0 0;0 m1 0;0 0 m1]);

    f = [0;m1*g;0];

    gamma = [-2*x1dot^2-2*y1dot^2;0];

    qddot = invM*(f-Cq'*inv(Cq*invM*Cq')*(Cq*invM*f-gamma));

    lamda = inv(Cq*invM*Cq')*(Cq*invM*f-gamma);

    dqdt = [qp(4:6);qddot];
end
```

A.2.d. Simple Pendulum Projection Function, Baumgarte Stabilization

```
function dqdt = Pendulum_ProjectionBaum(t,qpb) % Single Pendulum
% Projection Method with Baumgarte Stabilization

    global g m1 alpha beta
    % Read current state
    x1 = qpb(1);
    y1 = qpb(2);
    z = qpb(3);
    x1dot = qpb(4);
    y1dot = qpb(5);
    zdot = qpb(6);

    C = [x1^2+y1^2-1;z];
    Cq = [2*x1 2*y1 0;0 0 1];
    C_dot = Cq*qpb(4:6);

    % Baumgarte Stabilization
    B = 2*alpha*C_dot + beta^2*C;

    %% Update acceleration
    invM = inv([m1 0 0;0 m1 0;0 0 m1]);

    f = [0;m1*g;0];

    gamma = [-2*x1dot^2-2*y1dot^2;0]-B;

    qddot = invM*(f-Cq'*inv(Cq*invM*Cq')*(Cq*invM*f-gamma));
```

```
    lamda = inv(Cq*invM*Cq')*(Cq*invM*f-gamma);

    dqdt = [qpb(4:6);qddot];
end
```

A.2.e. Simple Pendulum Goicolea and Orden Function

```
function dqdt = Pendulum_GOPenaltyFunc(t,qg) % Single Pendulum
% Goicolea and Garcia-Orden Penalty Method
    constraintTolerance = 1e-10; % Tolerance for displacement and velocity
constraints
    global g m1 mu
    % Read current state
    x1 = qg(1);
    y1 = qg(2);
    z = qg(3);
    x1dot = qg(4);
    y1dot = qg(5);
    zdot = qg(6);

    C = [x1^2+y1^2-1;z];
    Cq = [2*x1 2*y1 0;0 0 1];

    %% Update acceleration
    M = [m1 0 0;0 m1 0;0 0 m1];

    f = [0;m1*g;0];

    qddot = inv(M)*(f-mu*Cq'*C);

    dqdt = [qg(4:6);qddot];
end
```

A.2.f. Simple Pendulum Index-2 Penalty Function

```
function dqdt = Pendulum_Index2Penalty(t,qi2) % Single Pendulum
% Index-2 Penalty Method
constraintTolerance = 1e-10; % Tolerance for displacement and velocity
constraints
global g m1 mu
    % Read current state
    x1 = qi2(1);
    y1 = qi2(2);
    z = qi2(3);
    x1dot = qi2(4);
    y1dot = qi2(5);
    zdot = qi2(6);

    C = [x1^2+y1^2-1;z];
    Cq = [2*x1 2*y1 0;0 0 1];

    qdot = qi2(4:6);
```

```
        %% Correct displacement constraint
    while abs(C) > constraintTolerance

        Q_delta = -Cq'*inv(Cq*Cq')*C;

        for i = 1:3
            qi2(i) = qi2(i) + Q_delta(i);
        end

        % Update displacement state
        x1=qi2(1); y1 = qi2(2); z = qi2(3);

    C = [x1^2+y1^2-1;0];
    Cq = [2*x1 2*y1 0;0 0 1];
    end

    C_dot = Cq*qi2(4:6);

    %% Update acceleration

    y = qdot;

    M = [m1 0 0;0 m1 0;0 0 m1];

    gamma = [-2*x1dot^2-2*y1dot^2;0];

    f = [0;m1*g;0];

    qddot = inv(M+mu*(Cq'*Cq))*(f+mu*Cq'*gamma);
    qdot = inv(eye(3)+mu*(Cq'*Cq))*(y+mu*Cq'*C_dot);

    dqdt = [qi2(4:6);qddot];
end
```

A.2.g. Simple Pendulum Direct Integration Function

```
function dqdt = Pendulum_Direct(t,qe) % Single Pendulum
% Static Penalty Method with Baumgarte Stabilization
    global g
    % Read current state
    theta = qe(1);
    omega = qe(2);

    %% Update acceleration

    dqdt = zeros(2,1);
    dqdt(1)=qe(2);
    dqdt(2)=-(g*sin(qe(1)));

end
```

A.3. Scotch Mechanism

```
close all; clear all; clear global;
set(0,'defaultlinelinewidth',2);
% Scotch Mechanism

%% Parameters
% Simulation
tfinal = 4; tstep = 0.0001;

% Model
global g m1 m2 tp L b mu alpha beta
g = 386.4; % [in/s^2] gravitational constant
m1 = 0.5/g; % [slug] mass of rod
m2 = 5/g; % [slug] mass block
tp=0:tstep:tfinal;
L = 25; % [inches] length of rod
b = 0.05; % [inches] width of rod
mu = 10^5;
alpha = 20;
beta = 20;

% Initial Conditions
th1init = 1; x2init = L*cos(th1init);
th1dotinit = 0; x2dotinit = 0;

q0(1) = th1init;
q0(2) = x2init;
q0(3) = th1dotinit;
q0(4) = x2dotinit;


%% Projection Method
%for i = 1:100
tic
disp('Solution Method: Projection');
[t,qp]=ode23(@Scotch_ProjectionFunc,  tp, q0);
%timer1(i)=toc;
%end
%mean(timer1)

% Process and Plot Solution
figure;
subplot(1,2,1); plot(t,qp(:,1),'-',t,qp(:,2),'--');
xlabel('Time [s]'); ylabel('q'); grid on;
title('Projection: Displacements'); legend('Theta1','X2')
subplot(1,2,2); plot(t,qp(:,3),'-',t,qp(:,4),'--');
xlabel('Time [s]'); ylabel('qdot'); grid on;
title('Projection: Velocities'); legend('Theta1dot','X2dot')

%% Penalty Method
%for i = 1:100
tic
```

```
disp('Solution Method: Penalty');
[t,q]=ode23(@Scotch_PenaltyFunc,  tp, q0);
%timer2(i)=toc;
%end
%mean(timer2)

% Process and Plot Solution
figure;
subplot(1,2,1); plot(t,qp(:,1),'-',t,qp(:,2),'--');
xlabel('Time [s]'); ylabel('q'); grid on;
title('Penalty: Displacements'); legend('Theta1','X2')
subplot(1,2,2); plot(t,qp(:,3),'-',t,qp(:,4),'--');
xlabel('Time [s]'); ylabel('qdot'); grid on;
title('Penalty: Velocities'); legend('Theta1dot','X2dot')

diff = abs(q-qp);
figure;
subplot(2,2,1);plot(t,diff(:,1));title('Method Difference: Theta1');
xlabel('Time');ylabel('Difference');
subplot(2,2,2);plot(t,diff(:,2));title('Method Difference: X2');
xlabel('Time');ylabel('Difference');
subplot(2,2,3);plot(t,diff(:,3));title('Method Difference: Theta1dot');
xlabel('Time');ylabel('Difference');
subplot(2,2,4);plot(t,diff(:,4));title('Method Difference: X2dot');
xlabel('Time');ylabel('Difference');
```

A.3.a. Scotch Mechanism Penalty Function

```
function dqdt = Scotch_PenaltyFunc(t,q) % Slider Crank Mechanism

    global g m1 m2 L b mu alpha beta
    F = -sin(t*pi/4);
    % Read current state
    th1 = q(1);
    x2 = q(2);
    th1dot = q(3);
    x2dot = q(4);

    C = [x2-L*cos(th1)];
    Cq = [sin(th1)*L 1];
    C_dot = Cq*q(3:4);

    % Baumgarte Stabilization
    B = 2*alpha*C_dot + beta^2*C;

    %% Update acceleration
    M = [m1*(4*L^2+b^2)/12 0;0 m2];

    f = [-m1*g*L*cos(th1)/2;-F];

    gamma = [-cos(th1)*L*th1dot^2]-B;

    qddot = inv(M+mu*(Cq'*Cq))*(f+mu*Cq'*gamma);
```

```
    lamda = inv(Cq*inv(M)*Cq')*(Cq*inv(M)*f-gamma);

    dqdt = [q(3:4);qddot];
end
```

A.3.b. Scotch Mechanism Projection Function

```
function dqdt = Scotch_ProjectionFunc(t,qp) % Scotch Mechanism

    global g m1 m2 L b alpha beta
    F = -sin(t*pi/4);
    % Read current state
    th1 = qp(1);
    x2 = qp(2);
    th1dot = qp(3);
    x2dot = qp(4);

    C = [x2-L*cos(th1)];
    Cq = [sin(th1)*L 1];
    C_dot = Cq*qp(3:4);

    % Baumgarte Stabilization
    B = 2*alpha*C_dot + beta^2*C;

    %% Update acceleration
    invM = inv([m1*(4*L^2+b^2)/12 0;0 m2]);

    f = [-m1*g*L*cos(th1)/2;-F];

    gamma = [-cos(th1)*L*th1dot^2]-B;

    qddot = invM*(f-Cq'*inv(Cq*invM*Cq')*(Cq*invM*f-gamma));

    lamda = inv(Cq*invM*Cq')*(Cq*invM*f-gamma);

    dqdt = [qp(3:4);qddot];
end
```

A.4. Slider-Crank Mechanism

```
close all; clear all; clear global;
set(0,'defaultlinelinewidth',2);
% Slider-crank Mechanism

%% Parameters
% Simulation
tfinal = 1.8; tstep = 0.0001;

% Model
global g m1 m2 m3 tp L1 L2 alpha beta mu
```

```
g = 9.81; % [m/s^2] gravitational constant
m1 = 1; % [kg] mass of the driving bar
m2 = 1; % [kg] mass of the connector bar
m3 = 1; % [kg] mass of the piston
tp=0:tstep:tfinal;
L1 = 2;
L2 = 2;
mu = 10^5;
alpha = 20;
beta = 20;

% Initial Conditions
th1init = 3*pi/4; th2init = acos(-cos(th1init))-(th1init);
x3init = 2*sin(th1init)+2*sin(th1init+th2init); % [m] Length is 2 for each
beam
th1dotinit = 0; th2dotinit = 0; x3dotinit = 0; % [m/s]

q0(1) = th1init;
q0(2) = th2init;
q0(3) = x3init;
q0(4) = th1dotinit;
q0(5) = th2dotinit;
q0(6) = x3dotinit;

%% Projection Method
tic
disp('Solution Method: Projection');
[t,qp]=ode23(@SliderCrank_ProjectionFunc,  tp, q0);
toc

% Process and Plot Solution
figure;
subplot(1,2,1); plot(t,qp(:,1),'-',t,qp(:,2),'--',t,qp(:,3),':');
xlabel('Time [s]'); ylabel('q'); grid on;
title('Projection: Displacements'); legend('Theta1','Theta2','X3')
subplot(1,2,2); plot(t,qp(:,4),'-',t,qp(:,5),'--',t,qp(:,6),':');
xlabel('Time [s]'); ylabel('qdot'); grid on;
title('Projection: Velocities'); legend('Theta1dot','Theta2dot','X3dot')

%% Penalty Method
tic
disp('Solution Method: Penalty');
[t,q]=ode23(@SliderCrank_PenaltyFunc,  tp, q0);
toc

% Process and Plot Solution
figure;
subplot(1,2,1); plot(t,q(:,1),'-',t,q(:,2),'--',t,q(:,3),':');
xlabel('Time [s]'); ylabel('q'); grid on;
title('Penalty: Displacements'); legend('Theta1','Theta2','X3')
subplot(1,2,2); plot(t,q(:,4),'-',t,q(:,5),'--',t,q(:,6),':');
xlabel('Time [s]'); ylabel('qdot'); grid on;
title('Penalty: Velocities'); legend('Theta1dot','Theta2dot','X3dot')

diff = abs(q-qp);
```

```
figure;
subplot(2,3,1);plot(t,diff(:,1));title('Method Difference: Theta1');
xlabel('Time');ylabel('Difference');
subplot(2,3,2);plot(t,diff(:,2));title('Method Difference: Theta2');
xlabel('Time');ylabel('Difference');
subplot(2,3,3);plot(t,diff(:,3));title('Method Difference: X3');
xlabel('Time');ylabel('Difference');
subplot(2,3,4);plot(t,diff(:,4));title('Method Difference: Theta1dot');
xlabel('Time');ylabel('Difference');
subplot(2,3,5);plot(t,diff(:,5));title('Method Difference: Theta2dot');
xlabel('Time');ylabel('Difference');
subplot(2,3,6);plot(t,diff(:,6));title('Method Difference: X3dot');
xlabel('Time');ylabel('Difference');
```

A.4.a. Slider-Crank Mechanism Penalty Function

```
function dqdt = SliderCrank_PenaltyFunc(t,q) % Slider Crank Mechanism

    global g m1 m2 m3 L1 L2 alpha beta mu
    % Read current state
    th1 = q(1);
    th2 = q(2);
    x3 = q(3);
    th1dot = q(4);
    th2dot = q(5);
    x3dot = q(6);

    C = [L1*sin(th1)+L2*sin(th1+th2)-x3; -L1*cos(th1)-L2*cos(th1+th2)];
    Cq = [L1*cos(th1)+L2*cos(th1+th2) L2*cos(th1+th2) -1;
        L1*sin(th1)+L2*sin(th1+th2) L2*sin(th1+th2) 0];
    C_dot = Cq*q(4:6);

    % Baumgarte Stabilization
    B = 2*alpha*C_dot + beta^2*C;

    %% Update acceleration

   M =  [(4*m1*L1^2/12) 0 0;
       0 4*m2*L2^2/12 0;
       0 0 m3];

    f = [2*m2*sin(th2)*(th2dot^2+2*th1dot*th2dot)-
m2*g*(2*sin(th1)+sin(th1+th2))-m1*g*sin(th1);
        -2*m2*sin(th2)*th1dot^2-m2*g*sin(th1+th2);
        0];

    gamma =
[((2*sin(th1)+2*sin(th1+th2))*th1dot^2)+(2*sin(th1+th2)*(2*th1dot*th2dot+th2d
ot^2));
        (-(2*cos(th1)+2*cos(th1+th2))*th1dot^2)-
(2*cos(th1+th2)*(2*th1dot*th2dot+th2dot^2))]-B;

    qddot = inv(M+mu*Cq'*Cq)*(f+mu*Cq'*gamma);
```

```
    lamda = inv(Cq*inv(M)*Cq')*(Cq*inv(M)*f-gamma);

    dqdt = [q(4:6);qddot];
end
```

A.4.b. Slider-Crank Mechanism Projection Function

```
function dqdt = SliderCrank_ProjectionFunc(t,qp) % Slider Crank Mechanism

    global g m1 m2 m3 L1 L2 alpha beta
    % Read current state
    th1 = qp(1);
    th2 = qp(2);
    x3 = qp(3);
    th1dot = qp(4);
    th2dot = qp(5);
    x3dot = qp(6);

    C = [L1*sin(th1)+L2*sin(th1+th2)-x3; -L1*cos(th1)-L2*cos(th1+th2)];
    Cq = [L1*cos(th1)+L2*cos(th1+th2) L2*cos(th1+th2) -1;
        L1*sin(th1)+L2*sin(th1+th2) L2*sin(th1+th2) 0];
    C_dot = Cq*qp(4:6);

    % Baumgarte Stabilization
    B = 2*alpha*C_dot + beta^2*C;

    %% Update acceleration
    invM = inv([(4*m1*L1^2/12) 0 0;
       0 4*m2*L2^2/12 0;
       0 0 m3]);

    f = [2*m2*sin(th2)*(th2dot^2+2*th1dot*th2dot)-
m2*g*(2*sin(th1)+sin(th1+th2))-m1*g*sin(th1);
         -2*m2*sin(th2)*th1dot^2-m2*g*sin(th1+th2);
         0];

    gamma =
[((2*sin(th1)+2*sin(th1+th2))*th1dot^2)+(2*sin(th1+th2)*(2*th1dot*th2dot+th2d
ot^2));
         (-(2*cos(th1)+2*cos(th1+th2))*th1dot^2)-
(2*cos(th1+th2)*(2*th1dot*th2dot+th2dot^2))]-B;

    qddot = invM*(f-Cq'*inv(Cq*invM*Cq')*(Cq*invM*f-gamma));

    lamda = inv(Cq*invM*Cq')*(Cq*invM*f-gamma);

    dqdt = [qp(4:6);qddot];
end
```

VITA

Troy S. Newhart

Mechanical & Aerospace Engineering
Old Dominion University
214A Kaufman Hall, Norfolk, VA 23529

Email:
tsn5048@gmail.com

Education:

M.S. Aerospace Engineering, Old Dominion University, December 2019
B.S. Aerospace Engineering, The Pennsylvania State University, May 2015