THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

INVESTIGATION AND BENCHMARKING OF QUADRATIC OPTIMIZATION ALGORITHMS FOR THE STUDY OF CONTROL SYSTEMS

ADAM BAHR
SPRING 2014

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Constantino M. Lagoa
Professor of Electrical Engineering
Thesis Supervisor

John D. Mitchell
Professor of Electrical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

When studying practical control systems it is often useful to have a model of the system to for simulation and testing. However, many control systems can be extremely complex, which makes them difficult to simulate using an ordinary computer. In other cases one cannot determine the components of the control system individually; only the input and the output to the system are known. One solution to this issue is the use of optimization techniques to find a simpler model of the system.

Optimization is a process that seeks to find the best possible solution to a particular problem that can have an infinite number of valid solutions. In the case of control systems, there are typically many systems that can produce an input/output relationship that are similar to each other. When only an input and output is known, and not the transfer equation of the system itself, it is necessary to use optimization to find the most effective transfer function. With optimization it is possible to find a model of a system that has both a minimal number of poles and a relatively small error to the system being modeled, even if the original system has an extremely complex transfer function

Due to the nature in which the optimization function is derived, it is necessary to use quadratic optimization or quadratic programming, to develop a model of the system. Quadratic programming problems involve both a linear component and a quadratic component, and can be subject to one or many constraints. For the systems being analyzed in this study, the standard form of the quadratic program is:

$$\min_{x} \; x^T H x + \; f^T x$$

$$s.t. \; Ax \leq b$$

In this problem, x represents the coefficients of poles used in the system. Therefore the optimization process will attempt make as many entries of x equal to zero, while still keeping the

error in the system a relatively small number. All of the computations are done in the MATLAB environment, using publicly available solvers and toolboxes to streamline the testing. Each solver was tested for speed of solution and accuracy of the objective value, and if this was not attainable, the point at which the solver failed was determined.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Dr. Constantino Lagoa and Korkut Bekiroglu for their assistance throughout this project. Their guidance ranged from explaining foreign concepts to me at an understandable level to providing ideas on how to proceed when I had issues with the MATLAB code and solvers. With their insight, I was able to learn more about the study of control systems and more specifically quadratic programming. Additionally, I am grateful to Dr. Mitchell for being the second reader on this project. Finally, I would like to thank my parents for their continued support of all of my endeavors over the past four years

Chapter 1

Introduction

Control Systems

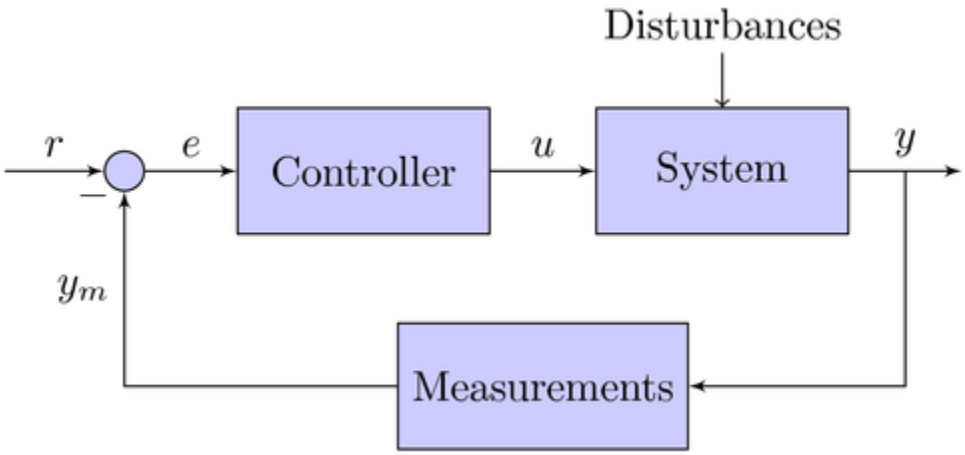
Control systems are used in many applications both in industry and in everyday life. All control systems have both an input and an output, and a governing function that regulates the relationship between the two, known as the transfer function. The user will set an input with the expectation that the system will create the desired output. In practical terms, the system must also account for noise both in the input and its internal connections, as well as external factors that could affect the desired performance of the system.

A simple example of a control system is the temperature setting on an oven. The temperature is set, and once that temperature is reached, the oven is expected to maintain that temperature throughout the entire cooking process. The temperature must remain constant even when the door is opened or food is placed in it, both examples of external factors that can affect the functionality of the system. A more complex example of a control system is the autopilot found in almost all large aircraft used today. The pilot sets the aircrafts desired heading, speed and altitude, and the control system positions the aircraft in the proper location. In this example the system must also account for wind that interacts with the aircraft and noise introduced in the electronic signals that are used in the control system itself (Mastascusa, 2014).

An important feature of practical control systems is the use of feedback to create a closed loop system. Feedback is data that gets sent back to the controller that describes the condition of the output. As seen in Figure 1-1, most control systems subtract the feedback signal from the input signal, giving the error. This is known as negative feedback. The goal in designing the

controller is to minimize the error signal (ideally it would be zero) for a specified range of inputs. If this occurs, the output function, y, would exactly match the input function, r, meaning the controller successfully controlled the system (Mastascusa, 2014).

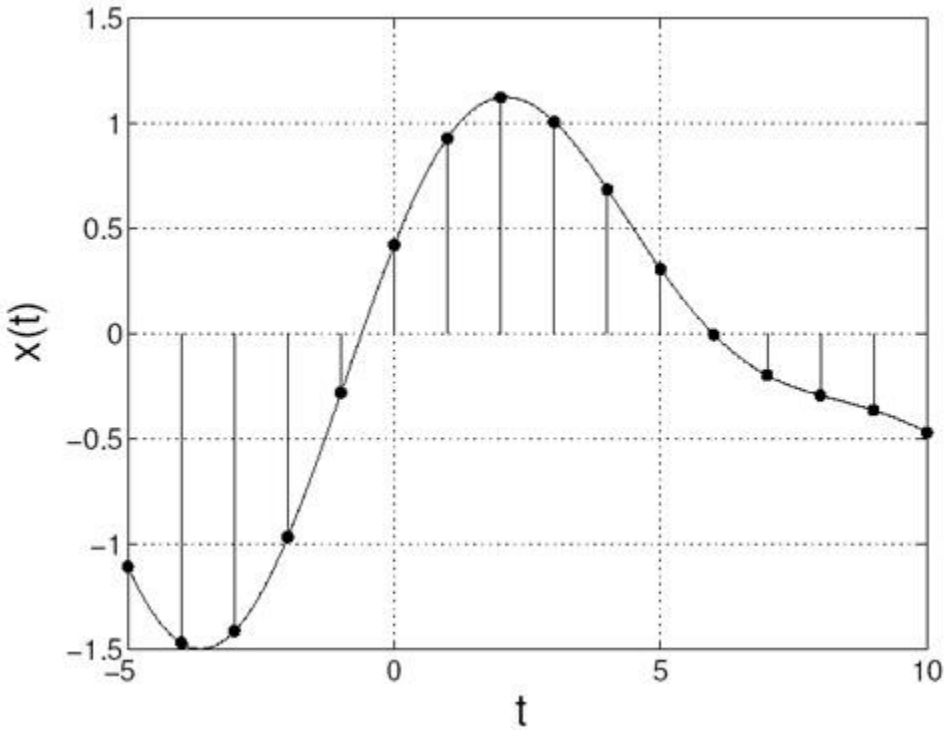
Figure 1-1: Block Diagram of a Simple Control System (Fauske, 2004)



When designing the controller, it is necessary to know what the model of the system transfer function is. The transfer function of the system must be determined in order for a controller to be created that will meet desired performance specifications. In the case of the autopilot, the controller is represented by a central computer that controls the heading, speed and elevation of the aircraft. By increasing fuel flow to the engines, or using a servo to move an aileron or rudder, the computer can alter the trajectory of the plane. The system to be modeled is the behavior of the entire airplane; its heading, pitch, roll, etc. Yet the transfer function of this system is difficult to determine from the components themselves. For this reason it is necessary to use quadratic programming to develop a model of these systems. Then a controller can be designed for the system model that will be able to reliably meet its performance specifications (Harris, 2007).

Controllers can be implemented using either a discrete time system or a continuous time system, and some can contain a combination of both. With the advent of cheaper and more powerful computers however, more controllers are being implemented with discrete systems. Like their names, discrete systems use discrete time signals while continuous systems use continuous time signals. The difference between the two types of signals is illustrated in Figure 1-2. The continuous signal is defined for every possible instance of time, while the discrete signal only exists at specific time intervals (in this example it is sampled at each second). The time intervals are determined by the sampling rate, which can be altered based on the needs of the controller (Gjendemsjø, 2004).

Figure 1-2: Difference between a Continuous Time Signal and its Discrete Counterpart (Gjendemsjø, 2004)



Stability

One of the most important properties of any real control system is its stability. Recalling the example of the oven control system, instability of that system would be catastrophic. For instance, if the user set the temperature to 400 °F he or she would expect it to remain there for the duration of the cooking time. But, if the action of opening the door caused the oven to increase the temperature without restraint, significant damage could be caused to anyone near the oven. Because of this, it is extremely important for control systems to be designed with stability, even (and especially) for inputs that are unexpected or unwanted.

The frequency domain of discrete time systems is known as the z domain, which is found by taking the z transform of the time domain function. A unique factor of frequency domain analysis is the ability to easily determine the stability of the system based on the location of the poles in the z plane. It can be shown that for a system to be stable in the time domain, each one of its poles must exist within the unit circle in the frequency domain. This allows for a relatively easy determination of system stability, once the z-transform of the system is determined.

Optimization

The goal of optimization is to find the highest or lowest value that a function can acquire when subject to a given set of constraints. The constraints act as bounds for this objective function, meaning the optimal value cannot be searched for across the entire domain, it must exist within the bounds of the constraints. Any point on the objective function that is within the bounds is known as a feasible point, with the optimal point being the lowest or highest feasible point in the domain. In the equations used in this study, the minimum value of the objective function will be sought after, rather than the maximum (Mathematical Programming, 2014). Optimization can be further sub classified based on the nature of the function that is being optimized. In this study, the objective function was quadratic in nature, while the constraints were linear.

Chapter 2

System Identification

Referring to the autopilot control system mentioned in the introduction, the system to be controlled is, as expected, extremely complex. However, it is relatively easy to measure the output of this system for any given input. Using this fact along with quadratic optimization methods, a simpler representation of a complex system like the autopilot can be determined.

As discussed previously, for any system to be stable, each pole must reside within the unit circle. This result means that for a stable system, the transfer function can be represented by a linear combination of the poles in the unit circle. In fact, the general form of any transfer function in the discrete time frequency domain can be written as:

$$\frac{Y(z)}{R(z)} = K\frac{\prod^{m}(z - z_i)}{\prod^{n}(z - p_i)}$$

Using partial fraction expansion, this can be reconfigured to yield:

$$\frac{Y(z)}{R(z)} = \frac{k_1 z}{z - p_1} + \cdots + \frac{k_n z}{z - p_n}$$

In this equation, the p_i's are the poles of the system and the k_n coefficients are the scaling factor for each pole (Phillips, Nagle, 1995). Thus, by generating a systematic "grid" of poles in the unit circle and scaling them by the proper factor, any system can be represented using these poles with only a small error. The grid of poles is generated in the MATLAB code and places the poles (referred to as grid modes in the MATLAB code) at a uniform distance apart in the complex plane. The MATLAB code used in this study can be found in Appendix A.

In the particular problem being solved in the study, the optimization function is created in a several step process. First, the output of the system to be modelled is measured. Next, an

additional output function is created using the linear combination of the first order transfer functions from each grid mode. These two outputs are subtracted from one another, yielding the error, and this error is one of the values to be minimized by the objective function. In addition, because it is desired to use as little poles as possible when representing the system, there is a "penalty" placed on the number of poles used. This penalty is implemented by taking the sum of the coefficients and adding this value to the objective function, effectively requiring it to be minimized as well.

The form of the optimization problem can be seen in below, with a description of each variable found in Table 2-1:

$$\min_{b,c} \ \|y - uMc\|_2^2 + \gamma \sum_{i=0}^{n} b_i$$

$$s.t. -b_i \leq c_i \leq b_i$$

Table 2-1. Description of Variables Used

Variable	Description
y	A 1xp vector of the values of the real system. P is the number of discrete time instances used in calculating the output (this value is 20 in the MATLAB code)
u	A 1xp vector comprised of a Gaussian distribution of numbers. This represents an input to the system which will excite all modes of the system
M	A pxn matrix of the impulse response of the grid modes
c	A 1xn vector of coefficients that scale each impulse response
γ	A scaling variable used to alter the relative importance of the number of poles used vs. the error of the simulated system
b	A 1xn vector of the absolute value of c
n	The number of grid modes used in the system

Chapter 3

Solvers

MATLAB, Toolboxes & Solvers

The MATLAB programming language was used to conduct the testing in this study. MATLAB is uniquely suited for this project because it provides relatively easy to use coding standards to create and work with large matrices. In addition, many QP solvers are written in MATLAB or have a version of the software that is compatible with the MATLAB interface. This allows the testing process to be streamlined, as the problem can be configured in a MATLAB .m file, then a specific solver function can be called from this.

MATLAB allows for the creation of solvers by third party software and these can provide additional functionality to the MATLAB environment. Solvers can be either written in the MATLAB language or they can be written in a different language such as C++ or FORTRAN. When the solvers are written in a code other than MATLAB, a certain steps must be taken before they can be used in the MATLAB code. First, a MATLAB file must be written by the software designer that uses libraries built into MATLAB to translate the original solver code. This translation is saved as a MATLAB executable (.mex) file. The MEX file is a binary file that acts as a MATLAB function file would, with the file name being the name used to call the solver in the code (MathWorks, 2014).

MATLAB also includes support for toolboxes that can further simplify the code generation process. Toolboxes are similar to functions in that they can be called from MATLAB code. However the toolboxes are usually much more complex than functions and can even

contain functions themselves. The particular toolbox used in this study was YALMIP, as it had support for a wide variety of quadratic solvers.

Toolboxes act as an intermediary between user created MATLAB code and the solvers. Although the solvers can be called directly from the MATLAB code, it can be useful to use a toolbox as a way to standardize the code. The inputs for a problem to be solved can be passed to the toolbox in a standard fashion, and in turn will re-work these inputs to be compatible with the particular solver the user wishes to use. This saves time for the user from having to learn the specific format of inputs for many different solvers and also reduces code used in the main program. However, toolboxes can also use large amounts of processing time in re-working the problem for each solver, which can be a hindrance as the size of the problem to be solved grows.

Standard Form

Using the YALMIP toolbox and the other solvers required the modification of the quadratic programming problem to put into a relatively standard form. Some of the solvers used a slight variation of the standard form, but for conciseness, only calculations used to work the problem into the YALMIP standard from will be discussed. The standard form of the quadratic programming equation used by YALMIP is:

$$\min_{x} \; x^T H x + f^T x$$

$$s.t. \; Ax \leq b$$

Using properties of the 2-norm for vectors, as well as other matrix properties, the original quadratic optimization problem can be transformed to match the form above. A more detailed operation and explanation of this transformation can be found in Appendix 2.

Solvers Studied

Below is a table of the MATLAB solvers that were tested and the type of optimization problem they were designed to solve. In order to find a baseline, some solvers that were not specific to quadratic programming were used. The results of these solvers were compared to the results found from the solvers that were specific to quadratic programming problems.

Table 3-1. Solvers Studied

Name	Type
OOQP	QP specific
qpOASES	QP specific
CLP	General- specialized for Mixed Integer
Quadprog	QP specific
SeDuMi	Symmetric Cones
SDPT3	Cones

Chapter 4

Results

To standardize the testing process, and to find useful differences between each solver, all tests were conducted on the same computer, running MATLAB R2013a. The specifications of the computer used for testing can be found in Table 4-1. Initial tests with the solvers and the YALMIP toolbox proved to be successful. In fact, when using less than 50 grid modes all of the solvers tested were effective. The results of each solver can be seen in Table 4-2 below. It is important to note that the objective value will change for different solvers. This is due to the fact that MATLAB generates a new set of random numbers for the input each time the code is run. Thus, each optimization problem is slightly different from the previous one.

Table 4-1. Computer Specifications

Operating System	Windows 7 Home Premium SP1 64-Bit
Processor	Intel Core i5 M 450
Clock Speed	2.4GHz
RAM	4.0GB

Table 4-2. Solver Results (36 Grid Modes)

Solver	Time (s)	Objective Value
Quadprog	1.22	1.4558
qpOASES	.55	1.5694
CLP	.386	1.5101
OOQP	.76	1.358
SeDuMi	1.869	1.4375
SDPT3	2.855	1.3758

It can be seen from Table 4-2 that CLP was the fastest solver, with the quadratic specific solvers taking the next three spots. As expected, the more general solvers SeDuMi and SDPT3 had the worst performance, taking the longest time to solve the optimization problem. Unfortunately, as the number of grid modes used increased, the solver time rose at an even faster rate. These larger problems also consumed more computer resources, using more than 1GB of RAM and more than 50% of the processing power for the duration of the solution calculation. Some of the solvers were not able to be fully tested, as they took more than 3 hours to find a value. The results of the test at 973 grid modes can be found in Table 4-3.

Table 4-3. Solver Results (973 Grid Modes)

Solver	Time	Objective Value
Quadprog	2hrs, 11min	0.741
qpOASES	1hr, 47min	0.8761
CLP	2hrs, 23min	0.6688
OOQP	2hrs, 39min	0.7776
SeDuMi	Processing time was > 3hrs	
SDPT3	Processing time was > 3hrs	

From Table 4-3, a similar pattern can be seen when compared to the 37 grid mode test of Table 4-2. Both of the non-quadratic solvers took an extremely long amount of time (greater than 3hrs) while the QP specific solvers generally performed better. Again, the outlier in both tests is the CLP solver, which for a non QP specific solver performed relatively well.

It is important to note that the objective values determined in the tables above do not reflect the values calculated in the MATLAB function code shown in Appendix A. This is due to the fact that a constant term is left out of the standard form optimization calculations in the MATLAB code. Thus, in order to find the true objective value for the original optimization function, this constant, must be added to the objective value calculated in MATLAB. The reasoning for the omission of the constant term is explained in more detail in Appendix B.

Chapter 5

Conclusion

The goal of this investigation was to study optimization problems for control systems and determine the effectiveness of quadratic programming solvers. Through repetitive tests and analysis, this goal was met. It was found that while the solvers work with extreme speed for smaller scale quadratic problems, the computational time dramatically increased as the number of poles used became larger. It is natural to expect that the solvers will take a longer time as the grid modes used increased, but it was found that this increase was not a linear relationship. It can be assumed that using a more powerful computer would have certainly decreased the amount of time each solver took. However, it can be inferred that as more and more grid modes are used (approaching 10,000 grid modes and above), the solving time for a supercomputer would be measured in hours, not minutes.

This investigation found that all of the optimization solvers tested required large amounts of processing time to compute an optimization problem with a large number of grid modes. Future algorithms could be developed that reduce the solving time specifically for large scale quadratic problems. Until these are developed however, the problems will have to be solved on a smaller scale, or a supercomputer will be necessary to solve the problems that cannot be reduced to a simpler form.

Appendix A

MATLAB Code

```
clear all
clc
close all


%% Gridding unit circle

% Approximate number of grid modes
ng = 1000;

n=round(sqrt(ng/pi)-1/2);

% Coordinates of the unit circle
r=1;
c=[0 0];

% Calculating the coordinates of each grid mode
  cg = zeros ( 2, ng );
  p = 0;
  for j = 0 : n
    i = 0;
    x = c(1);
    y = c(2) + r * 2 * j / ( 2 * n + 1 );
    p = p + 1;
    cg(1:2,p) = [ x,            y ]';
    if ( 0 < j )
      p = p + 1;
      cg(1:2,p) = [ x, 2 * c(2) - y ]';
    end
    while ( 1 )
      i = i + 1;
      x = c(1) + r * 2 * i / ( 2 * n + 1 );
      if ( r * r < ( x - c(1) ).^2 + ( y - c(2) ).^2 )
        break
      end
      p = p + 1;
      cg(1:2,p) = [            x,            y ]';
      p = p + 1;
      cg(1:2,p) = [ 2 * c(1) - x ,           y ]';
      if ( 0 < j )
        p = p + 1;
        cg(1:2,p) = [ x, 2 * c(2) - y ]';
        p = p + 1;
        cg(1:2,p) = [ 2 * c(1) - x , 2 * c(2) - y ]';
```

```
      end
    end
  end
grid_modes=(complex(real(cg(1,:)),cg(2,:)))';


%% Producing a test system to model

% Specify poles and zeros
num=[];
den=[-.4-.5i -.4+.5i];
indeces=find(grid_modes);
grid_modes=grid_modes(indeces);

% Produce system response
N=20;
t=0:N-1;
sys=tf(poly(num),poly(den),1);
u=[1 zeros(1,N-1)];
y=lsim(sys,u,t);
index = (0:N-1)';
scale = (ones(length(grid_modes),1)-abs(grid_modes).^2);


%% Solve optimization problem

% Construct the M matrix
M = zeros(N,length(grid_modes));
m = length(grid_modes);
for i=1:m
    a = dimpulse(poly([]),poly(grid_modes(i)),t+1);
    M(:,i) = a;
end


%% Translate the problem into the standard form

% Set the scaling factors
gamma = 1;
beta = 1;

% Find the number of poles being used
nh = length(grid_modes);

% Generate the random input signal
r = rand(N,1);
c = [r(1) zeros(1,N-1)];
u = toeplitz(r,c);

% Divide the M matrix into its real and imaginary parts
Mr = real(M);
Mj = imag(M);
```

```
% Calculations for entries in the H matrix
M1 = Mr'*(u')*u*Mr + beta*(Mj'*(u')*u*Mj);
M2 = -Mr'*(u')*u*Mj + beta*(Mj'*(u')*u*Mr);
M3 = -Mj'*(u')*u*Mr + beta*(Mr'*(u')*u*Mj);


%% Run the Solvers/ Toolboxes

% Put in the standard form
H = 2*[M1 M2 zeros(nh);M3 M1 zeros(nh);zeros(nh) zeros(nh) zeros(nh)];
f = [-2*y'*u*Mr 2*y'*u*Mj gamma*(ones(1,nh))]';
A = [eye(nh) zeros(nh) -eye(nh);-eye(nh) zeros(nh) -eye(nh);zeros(nh)
eye(nh) -eye(nh);zeros(nh) -eye(nh) -eye(nh)];
b = zeros(4*nh, 1);

% YALMIP
% The settings are used to call each individual solver
x = sdpvar(3*nh,1);
C = [A*x <= b];
O = .5*x'*H*x + f'*x;
settings = sdpsettings('solver','sedumi');
settings = sdpsettings('solver','quadprog','quadprog.MaxIter',10000);
settings = sdpsettings('solver','sdpt');
settings = sdpsettings('solver','clp');
settings = sdpsettings('solver','ooqp');
settings = sdpsettings('solver','qpoases');
sol = solvesdp(C,O,settings);
solution = double(x);

%OOQP
[status,x,gamm,phi,ky,z,lambda] = ooqp(f,H,[],[],[],[],A,[],b,'no');

% Quadprog
options = optimoptions('quadprog', 'MaxIter', 10000);
[x,fval,exitflag,output,lambda] =
quadprog(H,f,A,b,[],[],[],[],[],options);

% qpOASES
% This uses a slightly different standard form than YALMIP and other
% solvers, so it has separate calculations here
H = 2*[M1 M2 zeros(nh);M3 M1 zeros(nh);zeros(nh) zeros(nh) zeros(nh)];
g = [-2*y'*u*Mr 2*y'*u*Mj gamma*(ones(1,nh))]';
A = [eye(nh) zeros(nh) -eye(nh);-eye(nh) zeros(nh) -eye(nh);zeros(nh)
eye(nh) -eye(nh);zeros(nh) -eye(nh) -eye(nh)];
ubA = zeros(4*nh, 1);
[x,fval,exitflag,iter,lambda] = qpOASES(H,g,A,[],[],[],ubA);
```

Appendix B

QP Standard Form Transformation

The original form of the problem is:

$$\min_{b,c} \ \|y - uMc\|_2^2 + \gamma \sum_{i=0}^{n} b_i$$

$$s.t. -b_i \leq c_i \leq b_i$$

The M matrix will have both real and imaginary values because the time response is calculated for each individual pole. Therefore there will be no complex conjugated poles to produce a strictly real response. The imaginary part of the response is obviously not desirable, so this response will be placed in a separate 2-norm function to minimize it to zero. The equation becomes:

$$\min_{b,c} \ \|y - u(M_r c_r - M_i c_i)\|_2^2 \ + \ \beta \|u(M_r c_i + M_i c_r)\|_2^2 + \gamma \sum_{i=0}^{n} b_i$$

Using the properties of the 2-norm for vectors, the contents of both 2-norms yields:

$$(y - uM_r c_r + uM_i c_i)^T (y - uM_r c_r + uM_i c_i) + (uM_r c_i + uM_i c_r)^T (uM_r c_i + uM_i c_r)$$

$$\left(y^T - c_r^T M_r^T u^T + c_i^T M_i^T u^T\right)(y - uM_r c_r + uM_i c_i) + \left(c_i^T M_r^T u^T + c_r^T M_i^T u^T\right)(uM_r c_i + uM_i c_r)$$

After multiplying through and combining like terms the following result it found:

$$\begin{aligned} & c_r^T \left(M_r^T u^T u M_r + \beta M_i^T u^T u M_i\right) c_r + c_r^T \left(-M_r^T u^T u M_i + \beta M_i^T u^T u M_r\right) c_i \\ & \quad + c_i^T \left(-M_i^T u^T u M_r + \beta M_r^T u^T u M_i\right) c_r + c_i^T \left(M_i^T u^T u M_i + \beta M_r^T u^T u M_r\right) c_i \\ & \quad - 2y^T u M_r c_r + 2y^T u M_i c_i + y^T y \end{aligned}$$

To simplify the entry of the equation into the MATLAB code, the values in each of the parenthesis were stored as their own variable as shown:

$$M1 = \left(M_r^T u^T u M_r + \beta M_i^T u^T u M_i\right), \qquad M2 = \left(-M_r^T u^T u M_i + \beta M_i^T u^T u M_r\right),$$
$$M3 = \left(-M_i^T u^T u M_r + \beta M_r^T u^T u M_i\right)$$

The remaining b_i summation term can be put into matrix form simply by multiplying the b vector by a row vector of ones.

The original bounds also needed to be reformulated to conform to the standard form equation. With the addition of the complex component of c, the bounds become:

$$-b \le c_r \le b, \qquad -b \le c_i \le b$$

Separating the inequalities and resolving each to introduce a zero term yields four inequalities:

$$c_r - b \le 0, \qquad c_i - b \le 0, \qquad -c_r - b \le 0, \qquad -c_i - b \le 0$$

The standard form of the equation is:

$$\min_x \; x^T H x + f^T x$$
$$s.t. \; Ax \le b$$

Thus, using the expressions found above, the values for each variable in the standard form equation can be determined:

$$x = \begin{bmatrix} c_r \\ c_i \\ b \end{bmatrix}, \qquad H = \begin{bmatrix} M_1 & M_2 & 0 \\ M_3 & M_1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad f = \begin{bmatrix} -2y^T u M_r \\ 2y^T u M_i \\ 1 \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & 0 & -1 \\ -1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & -1 & -1 \end{bmatrix},$$

$$b = 0$$

It is important to note that the $y^T y$ term is omitted form the standard form equation. This term is a constant, so it will not affect the position of the optimization vector, but it will affect the objective value of the optimization function. Thus, the value of the $y^T y$ term was added to the objective value that was calculated using MATLAB. This is the result that is found in both Table 4-2 and Table 4-3.

BIBLIOGRAPHY

Boyd, Stephen P., and Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK: Cambridge UP, 2004. Print.

Dawkins, Paul. "Calculus I - Optimization. "*Pauls Online Notes*. Web. 18 Mar. 2014. <http://tutorial.math.lamar.edu/Classes/CalcI/Optimization.aspx>.

Fauske, Kjell M. "Example: Control System principles." *Control System Principles*. TEXample, 23 Nov. 2006. Web. 03 Mar. 2014. <http://www.texample.net/tikz/examples/control-system-principles/>

Gjendemsjø, Anders. Illustrations. OpenStax CNX, 9 Jan. 2004. Web. 05 Mar. 2014. <http://cnx.org/content/m11443/1.33/>

Harris, William. "How Autopilot Works. "*HowStuffWorks*. HowStuffWorks.com, 10 Oct. 2007. Web. 20 Mar. 2014. <http://science.howstuffworks.com/transport/flight/modern/autopilot3.htm>.

"Introducing MEX-Files." *Documentation Center*. MathWorks, n.d. Web. 20 Mar. 2014. <http://www.mathworks.com/help/matlab/matlab_external/introducing-mex-files.html>.

Mastascusa, E. J. "An Introduction To Control Systems." *An Introduction To Control Systems*. Bucknell University, Web. 04 Mar. 2014. <http://www.facstaff.bucknell.edu/mastascu/econtrolhtml/intro/intro1.html>.

Petersen, J.a.m., and M. Bodson. "Constrained Quadratic Programming Techniques for Control Allocation." *IEEE Transactions on Control Systems Technology* 14.1 (2006): 91-98. Print.

Phillips, Charles L., and H. Troy Nagle. *Digital Control System Analysis and Design*. Englewood Cliffs, NJ: Prentice Hall, 1995. Print.

"The Nature of Mathematical Programming. "*Mathematical Programming Glossary*. INFORMS Computing Society, Web. 18 Mar. 2014. <http://glossary.computing.society.informs.org/index.php?page=nature.html>

ACADEMIC VITA

Adam Bahr
amb6125@gmail.com

Education

- The Pennsylvania State University—Schreyer Honors College
- Graduation date: May 2014
- Major: Electrical Engineering
- Relevant Courses: Discrete and Continuous Linear Systems Analysis, Engineering Design, Electromagnetics, Circuit Analysis and Design, Communication Systems, C++ Programming

Professional Experience

Lutron Electronics – Coopersburg, PA *05/2013 – 08/2013*

- Upgraded LED light control device to use a different microcontroller
- Ported the existing software from the STM32F103 to the STM32F051 microcontroller

L-3 Communications – Williamsport, PA *05/2012 – 08/2012*

- Designed and built custom circuits to control high voltage power supplies
- Assisted in the diagnosis and repair of Joint Threat Emitter transmitters
- Diagnosed and repaired control boards for high voltage power supplies
- Wired and assembled fuse boxes, power supplies and a cooling system for a large transmitter

Activities

Penn State Food for Thought *09/2012 – Present*

- Raise funds to purchase food supplements for impoverished children
- Host an annual 5k Run and Date Auction

St. John Neumann Odyssey of the Mind Team Member *10/2001 – 05/2010*

- Designed and built many types of vehicles including hovercrafts, a "bomb-dropping" airplane and a 300-lb vehicle powered by a small R/C car battery
- Created a remote control car from scrap materials that became an airplane
- Made various electro-mechanical systems such as a frisbee thrower, tennis ball launcher, and a mini-elevator
- Placed 10^{th}, 6^{th}, 4^{th}, 3^{rd}, 3^{rd} and 2^{nd} at the World Competition

Technical Skills

- C, C++, MATLAB and LabVIEW Programming
- Microcontroller Programming and Implementation
- SolidWorks and Multisim/Ultiboard