MODELING AND CHARACTERIZATION OF SMALL UNMANNED AERIAL SYSTEMS (UAS)

An Undergraduate Research Scholars Thesis

by

PREETAM PALCHURU

Submitted to the Undergraduate Research Scholars program
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Research Advisor: Dr. John Valasek

May 2016

Major: Aerospace Engineering

TABLE OF CONTENTS

ABSTRACT

Modeling and Characterization of Small Unmanned Aerial Systems (UAS)

Preetam Palchuru
Department of Aerospace Engineering
Texas A&M University

Research Advisor: Dr. John Valasek
Department of Aerospace Engineering

The objective of this research aims to develop efficient methods to verify and validate linear time-invariant models derived from nonlinear time-variant systems. The parametric model will be compared to a non-parametric model of a small UAS (obtained from flight test data) totest for accuracy. If the parametric model is not as accurate as the non-parametric flight test model, then the parametric model will be tweaked to match the non-parametric model as closely as possible. A proper understanding of aircraft dynamical modeling is necessary to ensure proper adjustment of the model. In order for the parametric model to be accurate, the model must undergovalidation and verification. The physics of the plane should be validated. In other words, does the plane adhere to physical laws in flight? Once the model is validated, it must be verified. In order to deal with the nonlinearity of the problem, genetic algorithms will be used to match the eigenvalues of the parametric and non-parametric models as closely as possible.

DEDICATION

I would like to dedicate this paper to my parents for always pushing me to do my best work. I would not be where I am without their continual love and support.

ACKNOWLEDGMENTS

I'd like to thank Dr. Valasek for his tireless effort and patience in mentoring me. I've learned a tremendous amount and have had the opportunity to be a part of some fascinating projects in the time that I have worked with him. Not many get a chance to work with the brightest and most successful professors in their field. I am glad to say that I have gotten that chance.

CHAPTER I

INTRODUCTION

Parametric vs. non-parametric models

Parametric models of dynamical systems are important to analyze characteristics and obtain valuable information on system behavior under different situations/constraints. A parametric model that is as accurate as a non-parametric model is an extremely powerful tool for analysis and design. Non-parametric models can be obtained empirically and provides information about a specfiic test with specific constraints. However, it is extremely accurate. For this research the parametric model is created using a commercial software tool called Advanced Aircraft Analysis in which aerodynamic and geometrical properties of the aircraft can be easily adjusted using this software tool. The non-parametric model is obtained from nonlinear flight test data and then linearized. The linearized equations of motion are derived from Newton's Laws and are used to analyze a model to ensure its physics are proper. The state-space equations are also derived from first principles and put in matrix form. Each parameter coefficient is a dimensional stability or control derivative. Note that the non-dimensional stability and control derivatives are used to calculate the dimensional stability and control derivatives.

Equations of motion and state-space equations

The Equations of motion are shown in Equation sets 1 and 2 and State-Space Equations are shown in Fig 1.1. The A and B matrices in the State-Space Equations of Motion are extremely important. As mentioned above, the A and B matrices consist of dimensional derivatives that give information on an aircraft's flight characteristics and response to control input. These matrices

allow us to predict how an aircraft will perform in the air even before it flies. The A and B matrices are the center of this research problem.

$$m(\dot{U} - VR + WQ) = mg_x + F_{Ax} + F_{Tx} \tag{I.1}$$

$$m(\dot{V} + UR - WP) = mg_y + F_{Ay} + F_{Ty} \tag{I.2}$$

$$m(\dot{W} - UQ + VP) = mg_z + F_{Az} + F_{Tz} \tag{I.3}$$

Equation set 1: Drag, Sideforce and Lift Equations (respectively).

$$I_{xx}\dot{P} - I_{xz}\dot{R} - I_{xz}PQ + (I_{zz} - I_{yy})RQ = L_A + L_T \tag{I.4}$$

$$I_{yy}\dot{Q} + (I_{xx} - I_{zz}PR - I_{xz}(P^2 + R^2) = M_A + M_T \tag{I.5}$$

$$I_{zz}\dot{R} - I_{xz}\dot{P} + (I_{yy} + I_{xx})PQ + I_{xz}QR = N_A + N_T \tag{I.6}$$

Equation set 2: Rolling, Pitching, Yawing Moment Equations (respectively).

Below are the final form of the Equations of Motion represented in State-Space form. $\dot{x}$ is our longitudinal rate variables. $[A]$ is our matrix filled with dimensional stability derivatives and $[B]$ is filled with the dimensional control derivatives. x is the state vector and u is the control vector (in this case, it shows only one variable: the variable representing rudder deflection). A separate set of equations in state-space form can also be derived using the above equations to represent the Latitudinal/Directional direction of motion.

$$\dot{x} = [A]x + [B]u$$

Longitudinal State-Space Linear Equations of Motion

$$\begin{bmatrix} \dot{u} \\ \dot{\alpha} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} X_u' & X_\alpha' & X_q' & X_\theta' \\ Z_u' & Z_\alpha' & Z_q' & Z_\theta' \\ M_u' & M_\alpha' & M_q' & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ \alpha \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} X_{\delta_e}' \\ Z_{\delta_e}' \\ M_{\delta_e}' \\ 0 \end{bmatrix} [\delta_e]$$

Linear/Angular Velocity Notation

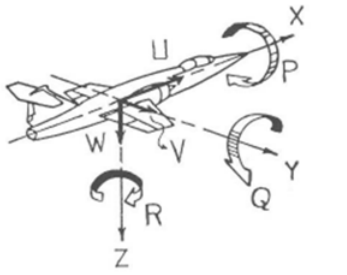

Fig. I.1.: State-Space Equations and Notations

Research issues

There are a few issues to overcome for this project:

1. Statistical databases do not exist for small UAS.
    (a) Statistical databases exist for larger (manned and unmanned) aircraft. However, small UAS do not have data readily available to use for analysis of UAS flight characteristics. Analysis methods for large vehicles do not scale well for small vehicles. Smaller aircraft have different aerodynamics and they operate at different Reynold's numbers. The same methods to analyze larger aircraft cannot be used to analyze smaller aircraft. Different methods must be developed and used to accurately bulid models of small UAS.
2. The small size of UAS results in
    (a) Smaller instrumentation and sensor volume. Smaller instruments tend to have less space, and are generally less accurate. When flight tests are run with these smaller sensors and instruments, it needs to be noted that the obtained values might not be as accurate as possible. This can cause some problems with accuracy and verification of the parametric model.
    (b) Lower instrumentation and sensor mass. The small UAS has payload limitations. Only so many instruments will fit inside the body of the UAS. This means that all the data necessary might not be collected on one single flight test (based on which data is needed, and which instruments can fit inside the UAS).
    (c) Less onboard power for instruments and sensors. Less power means that the UAS cannot support power-guzzling instruments. Doing so would drastically shorten flight time, and the shorter the flight time, the less accurate data that will be collected.
3. Restrictive sensor suite prevents all nevessary data from being measured, resulting less accurate measurements for those that can be measured (e.g. pitot static tubes). Some instrumentation won't be able to fit inside the UAS due to volume and weight limitations.

All of this leads to less accurate results. This means that special care must be taken when collecting data from flight tests. Data must be thoroughly analyzed to rule out ”bad” results.

4. Parametric models and identified models from flight data must be matched ad-hoc, i.e. by hand using engineering judgement.

    (a) Matching is done by comparing time histories, eigenvalues, eigenvectors, singular values, etc. Flight test data provides a benchmark to compare the parametric model to. Tweaking the parametric model to match the flight test data will ensure that the parametric model will accurately reflect the flight characteristics of the small UAS.

Vehicle description

Pegasus II is an unmanned aerial vehicle (UAV) that was designed and built in-house by the Vehicle Systems and Control Laboratory (VSCL) at Texas A&M University. Pegasus II is an improvement upon the design of the original Pegasus I (also built by VSCL). Pegasus uses a high wing and a twin-boom tail to help stability for flight testing. Pegasus often carries sensitive sensors and instrumentation, so stability in flight is key for obtaining accurate measurements and readings. Payload configuration on Pegasus is flexible. The tail and nose have ballast mounts, and the wings and main gear are adjustable on the fuselage. The wings and tail are built using a fiberglass/epoxy composite over a foam core with a carbon fiber spar. The fuselage is made of aluminum and fiberglass/epoxy composite. Pegasus has room for a substantial payload capacity. Pegasus flies at 400 ft and 50 knots when loitering. Takeoff weight is 95 lbs (including payload and fuel weight). Wing area is 18 ft^2. Wing span is 12 ft. Chord length is 1.5 ft.

CHAPTER II

METHODS

Software tools

Advanced Aircraft Analysis (AAA) is a software tool that uses a geometrical description of the plane to calculate stability and control derivatives (in non-dimensional form). AAA can also analyze other aspects of the plane, such as aerodynamics, range, and propulsion. However, for this project we are focusing on using AAA to obtain stability and control derivatives. Each non-dimensional derivative is calculated within its own page. These pages contain several input parameter cells for parameters such as aspect ratio, chord length, and various other geometrical values for the aircraft. AAA uses these input parameters and plugs them into specific equations to calculate the non-dimensional stability and control derivatives. Each of those input parameters can be changed (within reason) to make changes to the values of the non-dimensional derivatives. The model is modifyed in this fashion. This is a powerful tool because physical values of the plane can be ”modifyed” inside the model to change the derivatives and analyze flight characteristics of the plane without ever actually flying the plane.

These values are fed into a MATLAB script that calculates the dimensional forms of the stability and control derivatives. The dimensional derivatives are necessary to calculate the A and B matrices for the plane. The MATLAB script also calculates eigenvalues/eigenvectors and damping/frequency values for each eigenvalue. These eigenvalues and damping/frequency values are used to analyze different flight characteristics of the plane. Model verification requires comparisons of all these values. The linear and nonlinear models are analyzed side-by-side to make comparing the models more streamlined. A combination of comparing A and B matrices, eigenvalues, and time histories is necessary to verify the linear model.

MATLAB will also be used to build a Genetic Algorithm to run in order to optimize the A and B matrices of our parametric model so that the eigenvalues of the parametric model and the flight test model match closely. This will be explained in the ”Genetic Algorithms” section.

Model verification (ad-hoc method)

After developing a linear non-parametric model of Pegasus II, the parametric model (built in AAA) and the non-parametric model must be matched. The damping and frequency values for each mode must be compared and adjust the model in AAA to get the damping and frequency values to match those obtained from flight testing. Once all the damping and frequency values in the parametric model are matched with the values in the non-parametric model, the parametric model is accurate.

There are three ways to verify a linear model with a nonlinear model. First, the eigenvalues and eigenvectors of the aircraft's A and B matrices are compared. If there are large differences in the eigenvalues, then more than likely the linear model is inaccurate and needs adjusting. The second way is by comparing frequency ratios, damping ratios and time constants. Again, if there is a large discrepancy in the values the linear model is not accurate. The equations to calculate frequency and damping ratios are very specific. To match the frequency ratios of a linear model to the frequency ratios of a nonlinear model requires adjusting the proper non-dimensional coefficients to get values that correspond more closely with the nonlinear model values. The same is true for damping ratios and time constants. The third way is to compare time histories. Raw flight data is compared to the parametric model using the same control inputs from flight testing. This is done using MATLAB and the lsim function. Comparing time histories graphically shows how close the linear model matches the nonlinear flight test model.

After computation, the actual matching of the parametric model to the non-parametric model is done by hand. The parametric model is changed within Advanced Aircraft Analysis. Every time the model is modified slightly, the A and B matrices change slightly, and the aircraft dynamics change. The newly adjusted non-dimensional stability derivatives are plugged back into the MATLAB script to obtain new eigenvalues and time histories. If the models match closely, the process is finished. If not, the process repeats until the linear model and nonlinear model match within a specified error level using engineering judgement.

Comparing eigenvalues/eigenvectors and comparing frequency ratios/damping values are very similar in that the frequency ratios and damping values are derived from the eigenvectors. Comparing time histories is different from comparing eigenvectors. We use the $A\&B$ matrices to

obtain our eigenvalues/eigenvectors. Then we use eigenvalues to calculate the damping and frequency ratio values. Using all three comparisons increases the likelihood that there is a close match between parametric and non-parametric models. Sometimes one comparison gives different information than another comparison, so it is important to use all three concurrently.
It is important to keep in mind that the ad-hoc method of matching the parametric model to the flight test data is tedious and error-prone. There are multiple variables to address when matching the models ad-hoc. It's an iterative method that might not yield an accurate result. In order to overcome these shortcomings, algorithms can be employed to automate the verification process. Solving this research problem requires many iterations involving multiple variables. Using Genetic Algorithms provides an efficient way to iterate and find better solutions more quickly than an ad-hoc process.

Genetic algorithms

Overview

An algorithm is a series of steps for solving a problem. There are multiple types of algorithms and Genetic Algorithms are one type. A Genetic Algorithm is a problem solving method that is modeled after genetics and the idea of "survival of the fittest." Genetic Algorithms are most often used to solve problems where something needs to be optimized. Candidate solutions constitute the search space for the algorithm. Optimization problems that deal with easily enumerated numbers and linear spaces are not usually difficult to solve. However, when the search space is large or complicated, other techniques must be used. Brute computing power isn't feasible because generation of a solution would take too long. Genetic Algorithms are useful for complicated search spaces.
Genetic Algorithms start with a population of possible solutions. Each solution is represented through a "chromosome," which is an abstract representation of the solution. Each solution has its own "genes," which abstractly describe certain attributes of each solution. Devising these abstract representations for the solutions is not a straightforward process. In keeping with the analogy to nature, a Genetic Algorithm also includes rules on "reproduction" between chromosomes (solutions). Reproduction is used to perform mutations and recombinations over solutions.

Special care must be taken in creating the abstract representations of the solutions and reproduction rules since these representations can change the behavior of the Genetic Algorithm heavily.

Encoding

Each gene and chromosome is encoded by bit strings. Encoding is the process of representing individual genes. This process can use bits, numbers, arrays, or other objects. The type of encoding depends largely on the type of problem at hand. The most common form of encoding is binary encoding. Each chromosome encodes a binary string filled with bits. Each bit in the binary string can represent some characteristics of the solution. Binary encoding gives many different chromosomes with a small number of ”alleles.” This type of encoding is used in this research.

Fitness

At each generation, the algorithm selects the ”fittest” individuals to carry to the next stage. This is done using a fitness function. Each solution has a value for how ”fit” it is. This fitness is determined by the nature of the problem. In optimization problems, fitness is often measured in how ”optimal” the solution is by checking it against some true value or such. For certain problems, optimization means minimization/maximization. For these types of problems, a simple cost function can be used as the fitness function. For most other optimization problems, the fitness function is very dependent on the nature of the problem itself.

Genetic algorithm steps

Before the Genetic Algorithm can be run successfully, it requires four parts: Initial population of solutions(generated randomly or otherwise), solution representations, reproduction rules and the fitness function. With those four necessary parts, the algorithm iterates through the following four steps until the algorithm hits the stop criteria (which can be a tolerance for the fitness function, number of iterations, etc):

1. SELECTION: select individuals for ”reproduction.” The selection of solutions (or individuals) to cross together can be simple or complicated. Depending on the problem, the proper selection process must be chosen.
2. REPRODUCTION: Offspring are ”bred” by the selected individuals. The algorithm now generates new solutions based on the reproduction rules outlined in the algorithm. Crossing parents creates new individuals (children/offspring). Crossing genes of individuals is done after selecting the parents. This process is also problem-specific. The proper crossing process must be chosen for the problem at hand.
3. EVALUATION: The fitness of the new solutions is evaluated. A fitness function is used to evaluate each ”bred” solution.
4. REPLACEMENT: Old solutions are thrown out and the process is restarted using the new population as the initial population.

To illustrate how this algorithm is translated to code, and eventually to our problem, here is an example using MATLAB:

```
lb = [1 21 2 12 1 5 1 7 1 30];
ub = [100 100 100 100 100 100 100 100 100 100];
opts = gaoptimset(...
    'PopulationSize', 150, ...
    'Generations', 500, ...
    'EliteCount', 10, ...
    'TolFun', 1e-4, ...
    'PlotFcns', @gaplotbestf);
rng(0, 'twister');
[xbest, fbest, exitflag] = ga(@cantileverVolume, 10, [], [], [], [], ...
    lb, ub, @cantileverConstraints, [1 2], opts);
display(xbest);
fprintf('\nCost function returned by ga = %g\n', fbest);
```

This problem optimizes values for a specific problem outlined by the functions `cantileverVolume` and `cantileverConstraints`. The `rng(0, 'twister')` line creates

random values to start with for our initial population, within the lower and upper bounds set by the first two lines. The next several lines set the options for our Genetic Algorithm, such as initial population size, number of generations (iterations) to run, and tolerances (to throw the `exitflag` and cause the iterations to terminate). The `ga` function returns three things: an array of values for `xbest` and `fbest` (between `lb` and `ub`) optimized at each index. Within the `ga` function, the four steps outlined above take place. This is a very tidy and neat MATLAB package. As long as the proper fitness functions and representations are applied, the algorithm does the bulk of the iterations within the `ga` function.

Genetic algorithms in this research

Currently, matching of the parametric model to flight test data is done somewhat ad-hoc. Parameter values are frequently modified to reach a better solution. However, it is difficult to keep track of the numerous variables that affect the A and B matrices and therefore the eigenvalues of the matrices. There is much room for error and information oversight. It can keep track of all the variables and parameters at once and find a better solution more quickly than an ad-hoc process. The solution population will be A matrices. Each solution will have a slightly different A matrix. The genes of each solution will represent portions of the A matrix. The dimensional derivatives inside the A matrix can be collected into connected groups. Each gene will be changed slightly between solutions. With a sufficiently diverse initial population, we can find a very optimal solution with the Genetic Algorithm. The exact abstract representation of the solutions and the reproduction rules is still under work. To calculate fitness of each solution, we will compare the eigenvalues of each possible solution with the eigenvalues taken from flight test data A and B matrices. The closer the solution eigenvalues are to the flight test eigenvalues, the more ”fit” the solution. Again, exact representation of the fitness function is still under work.

Spring-mass-damper system

To demosntrate that a Genetic Algorithm can be used to work with a complicated system such as aircraft dynamics, it is first applied to a system where validation and verification is straightforward. Validation and verification for a system as complicated as airplane dynamics is a

much larger and more involved task. For a (homogenous) spring mass damper system, the following equations govern the motion of a mass attached to a spring and dampener.

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{II.1}$$

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = 0 \tag{II.2}$$

Solving the differential equation gives us the following equation.

$$M^2 + \frac{c}{m}M + \frac{k}{m} = 0 \tag{II.3}$$

Using the quadratic formula, the equation can be solved to find the "roots." The equation can also be placed into State-Space form with $x_1 = x$ and $x_2 = \dot{x_1} = \dot{x}$:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = [A] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-k}{m} & \frac{-c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Every spring-mass-damper problem can be identified uniquely by the three constants c, k, and m. Every unique SMD problem can be characterized by the eigenvalue of the A matrix.

Using Genetic Algorithm with spring-mass-damper system

Assume a unique Spring-Mass-Damper problem with specific values for c, k, and m. This problem has a single solution, and there is one eigenvalue for the A matrix, λ_1. The genetic algorithm starts with an initial A matrix with randomly chosen values for each element in the matrix. At each iteration of the Genetic Algorithm (through crossovers and mutations), the A matrix produced by the algorithm gets closer and closer to the A matrix given originally. The fitness function is the eigenvalue of the A matrix within the Genetic Algorithm (λ_2). When $\lambda_1 = \lambda_2$, the algorithm stops

because it has solved the specific problem of finding the A matrix that corresponds with a specific eigenvalue (and hence specific values of c, k, and m). This will be proven using MATLAB.

CHAPTER III

CODE AND TESTING

Code and description

The code used for this program is outlined below:

```
%Spring-Mass-Damper Genetic Algorithm
%Matches eigenvalues to target matrix
FitnessFunction = @fitness2; %sets fitness function
numVars = 4; %number of values to optimize
X0 = [0 0 0 0]; %sets initial population

options = gaoptimset(...
    'PlotFcn',{@gaplotpareto,@gaplotscorediversity},...
    'Vectorized','off',...
    'InitialPopulation',X0,...
    'PopulationSize',500,...
    'MutationFcn',@mutationuniform,...
    'Display','iter',...
    'CrossoverFcn',@crossoversinglepoint,...
    'TolFun',1e-15,...
    'Generations',200,...
    'EliteCount',10); %sets options for GA

A = []; b = [];
Aeq = []; beq = [];
lb = 0; %lower bound for solutions
ub = 100; %upper bound for solutions

[x, fval,exitFlag,Output,Population,Score] = gamultiobj(FitnessFunction,...
    numVars,A,b,Aeq,beq,lb,ub,options);

target = target(); %sets target matrix
run('dispeigen.m') %displays eigenvalues and errors
```

The fitness function is as follows:

```
function y = fitness2(x)
A = target(); %calls target matrix to compare
B = [x(1) x(2);
    x(3) x(4)]; %B matrix changes every iteration
[~, eig_A] = eig(A); %eigenvalues of target matrix
[~, eig_B] = eig(B); %eigenvalues of B matrix
eig_one = abs(eig_A(1,1) - eig_B(1,1)); %difference in first eigenvalues
eig_two = abs(eig_A(2,2) - eig_B(2,2)); %difference in second eigenvalues
y(1) = eig_one; %Objective 1 for fitness function
y(2) = eig_two; %Objective 2 for fitness function
end
```

The maximum number of generations, the type of mutation function, and the type of crossover function are parameters for the Genetic Algorithm. This GA uses uniform mutation and single point crossover to get from the parent solutions to the children solutions. The Genetic Algorithm outlined in the above MATLAB code is also a multiobjective Genetic Algorithm. This type of GA optimizes multiple objectives (constraints) at once. This is useful when dealing with multiple eigenvalues. The objectives are realized within the fitness function. The fitness function has two objectives:

1. Minimize the difference between the first eigenvalues for the target matrix and the individual matrix (being processed by the GA).
2. Minimze the difference between the second eigenvalues for the target matrix and the individual matrix (being processed by the GA).

The GA takes an initial population (a matrix of zeroes). The input for a GA in MATLAB must be a row vector. In order to preserve all of the information in the matrix, the MATLAB script converts the two-dimensional matrix into a row vector by appending each row of the matrix to the first row. When outputting the results of the GA, the row vector is converted back to a two-dimensional matrix. The GA iterates on the initial population, mutating and creating new generations until it finds a solution that has a fitness value under the tolerance prescribed by the

function. The target matrix is accessible by both the script housing the genetic algorithm and the fitness function. When the stopping criteria are achieved, the GA returns a list of the individuals in the latest generation. These individuals are run through a separate script that finds the best solution. The GA script also calculates the eigenvalues of the best solution and compares this solution to the eigenvalues of the target matrix. Some examples are shown in the following section.

Testing

Three example test matrices are described below:

The matrix *A*:

$$\begin{bmatrix} 0 & 1 \\ 3 & 2 \end{bmatrix}$$

was used to test the algorithm's matching capabilities. *A* has eigenvalues λ = -1,3. The GA returned 70 individuals, and the best individual is selected:

$$\begin{bmatrix} 0 & -0.3725 \\ -7.2522 & 2.1185 \end{bmatrix}$$

with eigenvalues λ = -0.896, 3.015. The errors for the eigenvalue calculations are 11.6% and .48% respectively.

The matrix B:

$$\begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix}$$

was used to test the algorithm's matching capabilities. *B* has eigenvalues λ = -0.7016, 5.7016. The GA returned 70 individuals, and the best individual is selected:

$$\begin{bmatrix} 0 & 7.9571 \\ 0.4989 & 4.9281 \end{bmatrix}$$

with eigenvalues λ = -0.7047, 5.6328. The errors for the eigenvalue calculations are .44% and 1.22% respectively.

The matrix C:

$$\begin{bmatrix} 0 & 1 \\ 8 & 3 \end{bmatrix}$$

was used to test the algorithm's matching capabilities. *C* has eigenvalues λ = -1.7016, 4.7016. The GA returned 70 individuals, and the best individual is selected:

$$\begin{bmatrix} 1.0035 & -1.1044 \\ -9.2973 & 1.9189 \end{bmatrix}$$

with eigenvalues λ = -1.7756,4.6981. The errors for the eigenvalue calculations are 4.17% and .07% respectively.

The matrix D:

$$\begin{bmatrix} 0 & 1 \\ 5 & -3 \end{bmatrix}$$

was used to test the algorithm's matching capabilities. *D* has eigenvalues λ = 1.1926, -4.1926. The GA returned 70 individuals, and the best individual is selected:

$$\begin{bmatrix} 1.0421 & 0.7155 \\ 0.7700 & -4.0797 \end{bmatrix}$$

with eigenvalues λ = 1.1476, -4.1851 The errors for the eigenvalue calculations are 3.92% and 0.18% respectively.

During testing trials, the eigenvalues had, on average, a 2.18% error from the actual eigenvalues.

Errors

While the eigenvalues may match, multiple matrices can have the same eigenvalue. In order to make this algorithm more robust, the fitness function must also include a check on the eigenvectors as well as the eigenvalues. This will provide another objective for the fitness function. With an added constraint, the solution will become more robust. Before this Genetic Algorithm can match eigenvalues for larger matrices, the algorithm must handle more objective functions in order to, not only match eigenvalues, but match actual matrix values.

Extension to State-Space [A] matrix

In order to extend this Genetic Algorithm to a 4x4 matrix like the *A* matrix used in the Longitudinal State-Space Linear Equations of Motion (see Introduction), the algorithm must compare the eigenvalues of the larger matrix to the target matrix. This requires the fitness function to consider more than just two eigenvalues that could be either real or imaginary. The GA would take an initial population that has 16 members in it (as opposed to the four in the spring-mass-damper problem). The multiobjective GA function can handle more than two objective functions to optimize. This is important when dealing with several eigenvalues. Extending this algorithm to a larger matrix will also allow for quick matching of a parametric model's *A* matrices to a non-parametric model's *A* matrices. This facet of the problem is very interesting and will be explored further.

CHAPTER IV

CONCLUSION

Parametric models of dynamical systems help analyze charactertics of the system behavior under certain circumstances. An accurate parametric model is a very powerful tool. If the parametric model is not as accurate as a non-parametric model (obtained from flight testing), then the parametric model must be matched to the non-parametric model. In order to verify and validate these two models, their respective stability coefficients should be compared. Planes that have similar longitudinal and lateral/directional A matrices tend to behave similarly during flight. An efficient method of verifying and validating these parametric models is to use a Genetic Algorithm. The Genetic Algorithm iterates through generations of solutions until it finds a solution that matches the non-parametric model. in order to prove that this technique can be used with nonlinear models, the Genetic Algorithm was tested on a spring-mass-damper system. The Genetic Algorithm worked very well in matching the eigenvalues of the target matrix with the matrix given in the starting population. This Genetic Algorithm has proven that it can match the eigenvalues of a target matrix with accuracy. However, the Genetic Algorithm must be modified to accurately match the actual values in the matrix and handle a larger matrix, so that it may be applied to airplane dynamics.

REFERENCES

[1] Digital Datcom. (n.d.). Retrieved August 1, 2015, from http://www.pdas.com/datcom.html

[2] Arthurs, F., Valasek, J., & Zeiger, M. (n.d.). Precision Onboard Small Sensor System for Unmanned Air Vehicle Testing and Control. *American Institute of Aeronautics and Astronautics*

[3] Lampton, Amanda. "Discretization And Approximation Methods For Reinforcement Learning Of Highly Reconfigurable Systems". Ph.D. Texas A&M University, 2009. Print.

[4] Sivanandam, S. N, and S. N Deepa. *Introduction To Genetic Algorithms*. Berlin: Springer, 2007. Print.

[5] Mathworks.com,. "Genetic Algorithm - MATLAB & Simulink". N.p., 2016. Web. 16 Feb. 2016.

[6] Wikipedia,. "Genetic Algorithm". Web. 17 Feb. 2016.

[7] Matthews, James. "Generation5 - A "Hello World!" Genetic Algorithm Example". *Generation5.org*. N.p., 2003. Web. 16 Feb. 2016.