

Fall 2021 ME459 Final Project Report  
University of Wisconsin-Madison

# Iterative Stability Searcher for Modeled Standing Balance

Jackson Fox

December 23, 2021

## Abstract

As part of the overall study of human motor control, a commonly targeted area of investigation is standing balance. As standing is an inherently unstable task, looking at the control methods and patterns that people use to maintain balance can provide insight into the approach the brain takes to motor control overall. Within standing balance, the Neuromuscular Coordination Laboratory at Wisconsin has identified a repeated pattern of behavior of the intercept in various force vectors on the foot as measured throughout a period of normal standing called the ZIP. To help understand if the identified ZIP behavior is a passive result of human anatomy or is an active optimization, an understanding is needed of ZIP behavior in all theoretical stable standing positions. To simulate this range of possible configurations, the standing human is modeled as an inverted two-segment pendulum with joint control.

This project was developed to identify configurations and joint torques of the two-segment inverted pendulum model that satisfy a set stability criteria. The behavior of the double inverted pendulum was modeled and analyzed to identify if configurations could be considered stable, with stability being defined as there occurring a point in the simulated motion where both the angle of both segments were within 0.1 degrees of 0, essentially vertical. The inverted double pendulum equations of motion were derived, represented and solved in MATLAB. To simulate the motion of the model under constant joint torques, the EOMs were parametrized and solved over a range of time using the ODE45 function. In order to optimize the efficiency of the searching program to handle many iterations of joint torques, two analysis approaches were attempted. The first used ODE45 to solve across a full range of motion and then searched through the results, the second used ODE45 to solve individual time steps, check the output, and then repeat across the range of motion. Both approaches were compared for speed, and the first approach was found to be dramatically more efficient and implemented in the final solution.

The solution comprises of a set of MATLAB functions that take input values for initial joint position and number of torque iteration steps, and over a user-input number of torque steps (each step being 0.1 Nm) the program determines if the configuration is considered stable or not based on the simulated motion. All stable configurations for the input values are captured and output to a file and a summary of the process is displayed to the user. The source code can be found in the git repo here:

<https://euler-login-2.wacc.wisc.edu/jmfox8/me459-final-project>.

The next step with this solution is to expand the iterative qualities to walk through multiple joint positions in addition to torque values to more completely capture the behavior across the entire state space. To continue the overall investigation, the output of the expanded searching programs will be used to identify the ZIP behavior of the “stable” configurations and comparing to experimental data.

## Contents

1. Problem statement.....	4
2. Solution description .....	5
3. Overview of results. Demonstration of your project .....	7
4. Deliverables: .....	9
5. Conclusions and Future Work .....	12
References.....	12

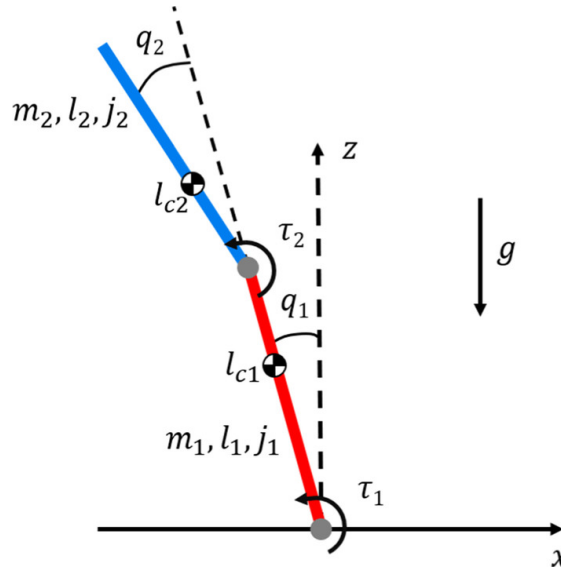
## 1. General information

- Department of Mechanical Engineering
- PhD Student
- Jackson Fox (Team Leader)
- I am not interested in releasing my code as open-source code.

## 2. Problem statement

The Neuromuscular Control Laboratory under Dr. Gruben focuses on understanding human motor control. An important area of focus is on the control of standing balance, and the lab has identified a metric in standing balance reflecting the intersection point of the various force vectors on the foot as measured throughout a period of normal standing through investigation of human standing balance studies. This metric is called the ZIP (Z-Intersection Point) [1]. Though this metric has been identified across numerous data sets in various studies and populations, analysis is ongoing into what, if any, function ZIP optimization performs in standing balance.

To investigate if the experimentally captured ZIP behavior is a positional value that is actively achieved through motor control (as opposed to a passive result of the geometry of human anatomy), the lab hopes to identify ranges of anatomical configurations and joint torques that can be considered stable and identify which of those configurations also demonstrate the documented ZIP behavior. A standing human is modeled using a two-segment inverted pendulum, which has been found to be the simplest model is able to replicate documented behavior utilizing simple feedback control [2]. The double inverted model, with identified parameters, can be seen in Figure 1.



**Figure 1:** Double Inverted Pendulum with Parameters [2]

The motion of the double inverted pendulum can be defined in as a series of two non-linear Ordinary Differential Equations, where the dependent variables are the joint angles of segment 1 and 2, representing the ankle and hip, and their respective derivatives for angular velocity and acceleration. The EOMs can be found as Equations 1 and 2 in the Appendix.

The goal of this project is to develop a program to identify “stable” configurations of this model, with stable defined as a system where both segments pass through vertical near simultaneously from a non-vertical starting position, through solving the EOMs across the comprehensive state-space defined by several constraints. Once identified, the ZIP behavior in these configurations can then be calculated and compared to the documented ZIP behavior in human subjects, to identify how common the documented ZIP behavior is across all potential stable balance positions.

### 3. Solution description

The solution to solve the EOMs and identify stability of the positions is written in MATLAB. The project was initially conceived and proposed as a project in C, with a focus on speed of execution that can be achieved using a compiled language vs. an interpreted language. As the state-space that this project wishes to search is quite large and could involve hundreds of thousands or millions of iterations, speed of execution is highly important. However, in order to solve the EOMs that define the system, a numerical approach to find the solution to the non-linear ODEs is required. C standard libraries do not have functions equipped to solve a series of ODEs. There are open-source C libraries that can calculate solutions to such problems, such as SUNDIAL (thanks to Dr. Negrut for sharing), however due to the complexity of implementing the library and lack of experience of the author, implementing the library was deemed infeasible for the scope of this project. MATLAB was selected as the replacement as it is designed around efficient and optimized numerical operations and contains numerous pre-built and optimized functions to solve series of ODE’s. The importance of speed of execution does not decrease with the change in language, hence two separate approaches were utilized for evaluating the ODE solutions at each configuration and identifying the stability said configurations. The results of the evaluation and impact are discussed later in this document.

The MATLAB solution is composed of two primary computational tasks – first is solving the EOMs that define the system for the position, velocity and acceleration of the model segments, specifically the angles (seen as “q” in Figure 1 above) through time for specific model configurations, and second is identifying if the resultant motion of the model satisfies the definition of “stable”. These tasks are accomplished with a set of MATLAB functions. The primary function takes input for initial conditions, holds definitions for all static parameters, and executes all other functions calls required for both solving the EOMs and identifying the stable configurations. To describe the solution, this section first details how the ODE45 function is utilized to solve the system of non-linear ODEs, then describes the two approaches used in the implementation of the ODE45 function.

The primary function used for the execution of the first computation task of solving the system of ODEs is the MATLAB function ODE45. ODE45 can efficiently calculate solutions over time for systems of ODEs but requires input in the form of  $\dot{q} = f(t, q)$  where q is the dependent variable. As seen in the Appendix, the EOMs need to be parametrized to this form in order to utilize ODE45 as there are numerous first derivatives with different associated terms. This requires the use of a Mass matrix to represent the terms associated with the second derivatives on the left side of the equation while isolating the terms connected to the dependent variable and first derivatives on the right. This allows the entire system to be represented as a system of four equations using the form  $M(t, q)\dot{q} = f(t, q)$  where q is a vector made up of the two model angle values and their first derivatives, parametrizing the second order system to a first. The different pieces of this equation were defined in separate functions, taking in a time value, initial angular information vector, and a 1x2 structure defining the static physical parameters of the model

(material properties of a representative human, taken from [2]). Utilizing the 1x2 structure allows the passing of extra parameters into functions that specify the number of inputs they can receive. This approach, coupled with the utilization of anonymous functions to parametrize the components of the Mass matrix and the input to the ODE 45 function, allow the complex system to be represented in a singular linear way that ODE45 can solve. The ODE45 function is modified using the “odeset” option ‘Mass’ to tell the function to process the solution utilizing the Mass matrix, and is executed using a set time input, the input EOMs as described above, and a set of initial conditions. Once ran, the function outputs a time vector and q vector, which captures the angular position and velocity of each joint across the time span that was input into the function.

As described in the problem statement, the purpose of this solution is to generate results for a significant range of input parameters. The parameters of value were the torques around both joints (assumed constant throughout the duration of the motion), the initial angular position and velocity of each joint, the timespan over which the solution was solved, and the step size used. For this project, the initial angles and angular velocity of the joints was set constant, and the results were captured for a range of torques. The initial torque values were captured from a function that determines the necessary torque to keep the model in static equilibrium for the determined initial position and velocity scaled by a certain factor, and then iterated with small increases in torque magnitude in a double nested for loop to explore the behavior of the model across a wide range of ankle and hip torques. The number of iterations of the double loop, which determines the speed of the program execution, and the initial joint angles are requested as a user input.

The second primary computational task in the solution is to identify if the result of the ODE45 solution and corresponding motion output satisfies the stable criteria. Per discussion with the Dr. Gruben, the criteria of “stable” was that the angle measures of both joints in the model needed to measure between -0.1 and 0.1 degrees at the same time, approximating a nearly perfect upright position. In order to identify the optimal approach when iterating over a very large data set, two methods were identified to check for stability.

The first method was to run the ODE45 solution over a set time range, then take the output position matrix and identify if at any time point within the motion the criteria was satisfied. This method simply looped through the columns of the output matrix, and if at any point in the loop identified that the entry for the ankle and hip position were both within the bounds, captured the torque values at that iteration of the torque loop and exited the stability loop, stepping into the next torque iteration loop.

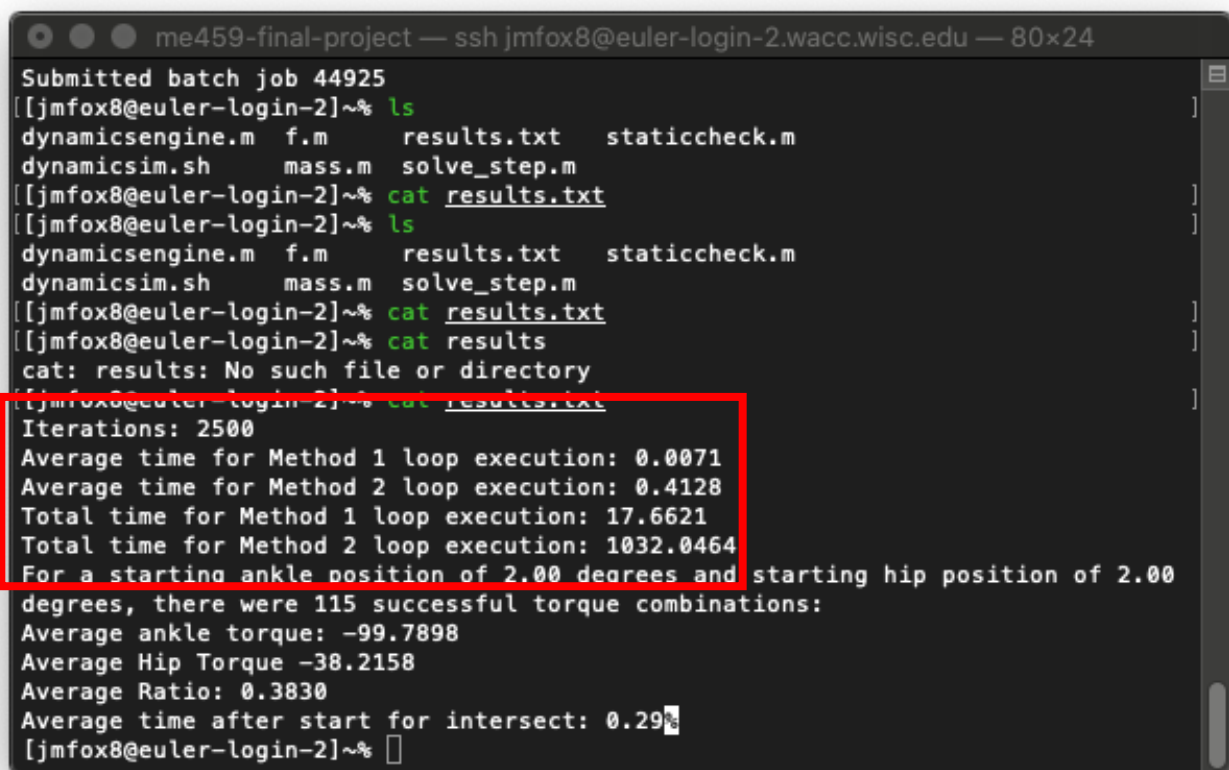
The second method was to run the ODE45 solution for individual time steps, the same as identified as inputs for the first method and check the output of ODE45 each time. Then, feed the output back into ODE45 as the input in a loop until the final output has covered the same span of time as identified as an input in the first approach. Both approaches were coded and executed – the results and takeaways are discussed in depth in Sections 4 and 7.

The final portion of the solution exists for analysis – the program captures and displays the average and total execution time per torque loop for each method, provide the number of successfully “stable” torque combinations that were identified, and the average values of those successful configurations. Finally, the program writes out to a .txt file all the successful torque combinations along with pertinent configuration information. If there are no successful stable configuration, the return informs the user as such and provides the search range over which the loops found no successful configuration

## 4. Overview of Results

The project was approached with two solution methods. The primary method for this submission utilizes a standalone function that takes inputs from the command line for initial angles and number of iterations, while also calling other MATLAB functions. A full description of all submitted files and their purpose can be found in Section 5.

There have been two primary outcomes of this project. First, the comparison of the two methods of execution, as detailed in section 3, has provided a clear result on the faster method. As seen below in a test output of the comparative program on Euler, the completion time per loop and overall, for a small execution size of 2500 (50 iterations of each for loop) was significantly slower for the individual step analysis method, by a factor of ~60.



```
me459-final-project — ssh jmfox8@euler-login-2.wacc.wisc.edu — 80x24
Submitted batch job 44925
[jmfox8@euler-login-2]~% ls
dynamicsengine.m  f.m      results.txt  staticcheck.m
dynamicsim.sh     mass.m   solve_step.m
[jmfox8@euler-login-2]~% cat results.txt
[jmfox8@euler-login-2]~% ls
dynamicsengine.m  f.m      results.txt  staticcheck.m
dynamicsim.sh     mass.m   solve_step.m
[jmfox8@euler-login-2]~% cat results.txt
[jmfox8@euler-login-2]~% cat results
cat: results: No such file or directory
[jmfox8@euler-login-2]~% cat results.txt
Iterations: 2500
Average time for Method 1 loop execution: 0.0071
Average time for Method 2 loop execution: 0.4128
Total time for Method 1 loop execution: 17.6621
Total time for Method 2 loop execution: 1032.0464
For a starting ankle position of 2.00 degrees and starting hip position of 2.00
degrees, there were 115 successful torque combinations:
Average ankle torque: -99.7898
Average Hip Torque -38.2158
Average Ratio: 0.3830
Average time after start for intersect: 0.29%
[jmfox8@euler-login-2]~% 
```

**Figure 2:** Test result demonstrating speed difference of different approaches

This dramatic slowdown is due to the repeated calling of the ODE45 function – where approach one only calls the function once per iteration, approach two can call it up to 100 times, which is computationally heavy. In order to be able to demonstrate this difference, I have included two versions of the driving program, one with the just the first approach (the actual final product) and one containing both approaches.

The second outcome of this project is a working solution that can identify stable configurations, as described thus far in the report, across a wide range of torque values in a reasonable amount of time. Upon execution of the dynamics engine function, the output of the program provides summary level information

for high-level analysis of results as well as a detailed list of the successful configurations. Information provided to the user in the direct output informs on the number of stable configurations as well as providing average values for those stable configurations, see examples in Figures 3 and 4 below.

```
[jmfox8@euler-login-2]~% cat results.txt
Iterations: 5625
Average time for Method 1 loop execution: 0.0078 sec
Total time for Method 1 loop execution: 44.0706 sec
For a starting ankle position of 3.00 degrees and starting hip position of 3.00 degrees,
there were 149 successful torque combinations in the range of
Min Ankle torque -41.6385 Nm and Max Ankle torque -153.2348 Nm
Min Hip torque: -14.5981 Nm and Max Hip torque: -65.7292 Nm
Average ankle torque: -149.5287 Nm
Average Hip Torque -57.2464 Nm
Average Ratio: 0.3828
Average time after start for intersect: 0.29 sec
```

**Figure 3:** Output of executable with stable results

```
[jmfox8@euler-login-2]~% cat results.txt
Iterations: 100
Average time for Method 1 loop execution: 0.0233 sec
Total time for Method 1 loop execution: 2.3290 sec
For a starting ankle position of 10.00 degrees and starting hip position of 25.00 degrees
, there were no successful torque combinations. The range searched was from:
Min Ankle torque -169.8226 Nm to Max Ankle torque -595.3790 Nm
Min Hip torque: -80.1038 Nm to Max Hip torque: -227.2440 Nm
```

**Figure 4:** Output of executable without stable results

The output file contains a list of the individual stable torque configurations that can easily be saved for further analysis in other programs. An example of the output file is included in the project submission in the Repo.

There are limitations to the applicability of the program in its current state. The model as it currently stands iterates over torque values for specific initial angular position and velocity values. In order to do a comprehensive survey of the state space, iterations need to occur across the angular position and velocity spaces of the joints as well, vastly increasing the required number of iterations. The primary reason this has not been implemented is that, in order to not waste computing power, I want to validate this model with Dr. Gruben before executing any programs requiring millions of loops/significant Euler power and time. The model also does not have any built-in constraints against “wasteful” iterations. There are very specific windows in which the torques can generate successful motion for each initial position – the vast majority of torques do not work. The program will not stop calculating once the valid state-space has been passed through, and will continue to iterate through large areas of instability. A potential mitigation for this going forward is to run moderately large simulations with larger step sizes, identify seed points of potential stability, and fully iterate around those points – however to be able to fully analyze the system a comprehensive, if wasteful, brute force style search is necessary.



## 5. Deliverables

This project consists of the following items. The final report has been uploaded to the ME459 Canvas Site, and the rest of the items are stored in a Gitlab Repo found here:

<https://euler-login-2.wacc.wisc.edu/jmfox8/me459-final-project>

### Non-Code:

- Final Report – Posted to Canvas, also included in the Gitlab Repo
- Successes.txt – the output file from execution of the solution. Included in Repo as an example in case of execution errors of function on Euler

### Code – Solution Executables (in Repo)

- dynamics\_engine\_func.m – This is the final standalone function to execute the solution that can be executed from slurm script. Requires input of parameters from user. This is the primary deliverable, and the function that can be used for validation of the output in Euler. This function calls mass, f, and staticcheck.
- dynamics\_engine\_final.m – This is the final MATLAB function for executing the solution. This program calls the dynamics\_engine\_func and provides it with input entered into the MATLAB UI.
- dynamicsengine.m – This is the version of the solution that includes both approaches as defined in this report – included in a form not requiring user input to allow execution and testing on Euler to see variance in performance time.
- dynamicsim.sh – slurm script that executes the dynamics\_engine\_func function, and allows the input of user parameters for the joint angles and number of torque steps
- dynamicsim\_comp.sh – slurm script that executes the dynamicsengine.m program. This does not take in user input and executes both solution approaches for pre-set iterations and angle value.

### Code – Supporting Functions (in Repo)

- solve\_step.m – function that runs ODE45 for a single time step. Utilized in the secondary solution approach of checking each individual step along the path. Only required for dynamicsengine.m executable.
- mass.m – function that defines the mass matrix for input to ODE45 based on initial conditions provided to the function. Required for all executables to be ran.
- f.m – function that defines the right side of the parametrized ODE functions for ODE45 based on initial conditions provided to the function. Required for all executables to be ran.
- staticcheck.m – function that identifies the minimum joint torques required to keep the model in a static position based upon initial conditions provided to the function. Required for all executables to be ran.

### To execute the primary deliverable code on Euler:

1. Ensure that dynamics\_engine\_func.m, mass.m, f.m and staticcheck.m and dynamicsim.sh are present in the working directory
2. Modify the three input values in dynamicsim.sh as desired. The first is the ankle angle from vertical, the second is the hip angle from the lower body segment, and the third is the number of torque steps (IMPORTANT – this value squared is the number of loops the solution will execute,

consider before using very large numbers). The execution line of the slurm script should look like the following:

```
matlab -batch "dynamics_engine_func '3' '3' '75'"
```

3. The expected output is two files: “results.txt” which captures the high level information about the search that was conducted, and “successes.txt”. The file “successes.txt” will only be generated if the search algorithm was able to find stable configurations based on the input parameters, but “results.txt” will always generate. See Section 4 for output samples.

**To execute the secondary deliverable code (showing the comparison of the two approaches):**

1. Ensure that dynamicsengine.m, solve\_step.m, mass.m, f.m, staticcheck.m and dynamicsim\_comp.sh are present in the working directory
2. Execute the slurm script (IMPORTANT – this is coded for 50 loops to demonstrate the dramatic timing difference – may take up to 10 min to execute)
3. View the generated output – the comp\_results.txt should include timing values from both execution methods, in addition to the standard output information.

## **6. Use of ME459 knowledge**

**ME459 Topics:**

- Defining and utilizing structures
- Passing pointers vs. variables to functions
- Creating programs that call other separately defined functions
- Functions that use functions (anonymous functions used to parametrize for ODE45)
- Dynamic memory allocation for speed
- Structuring looping logic based upon column – major order, as in MATLAB
- Identifying causes of slowdown by utilizing timing functions (tic and toc in MATLAB)
- Optimization of performance by reducing calls to main memory/relying on data already in the cache
- Utilizing MATLAB as a standalone from zsh and executing in Euler

**Evidence of Decoder – MATLAB Interface**

When testing the secondary solution approach, solving across individual time steps and comparing results for stability, I encountered an issue where the results I was getting for each time step would quickly become exponentially too high. Upon investigation of the variables being generated and used in the function solve\_step, I realized that there was a discrepancy in how I was passing the initial value vector. Upon initialization of the loop, the vector was in radians, and the ODE45 within solve\_step expected radians but the output of solve\_step, and therefore the input for the next iteration, was in degrees. This resulted in ODE 45 using degrees in a radian calculation, hence the exponential increases in position across very small time steps, which did not make physical sense.

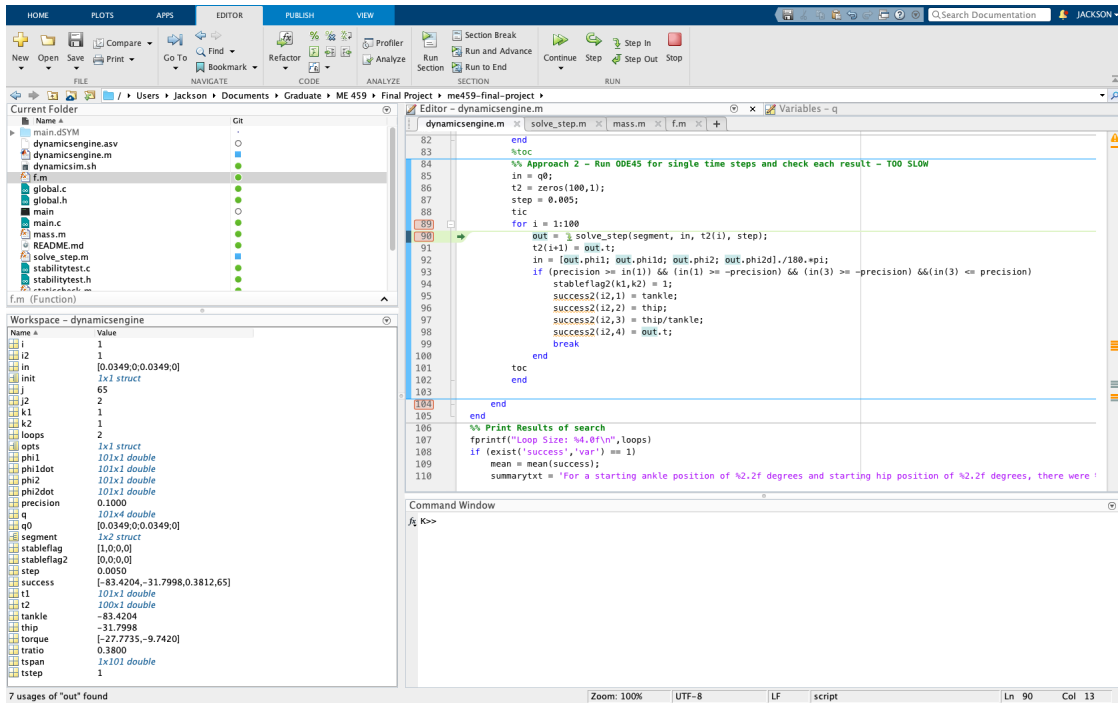


Figure 5: Main program interrupt to jump into solve\_step call

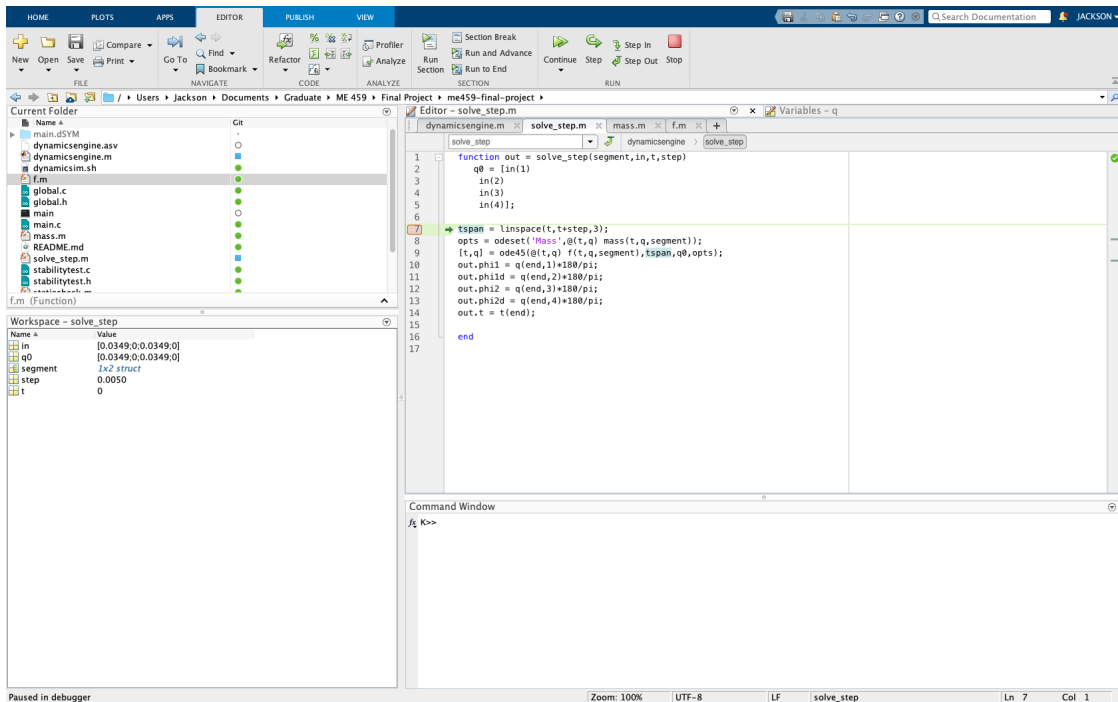


Figure 6: Solve\_step interrupt to identify issue

## 7. Conclusions and Future Work

Overall, this project was intended and succeeded as a proof of concept of the approach for identifying stable configurations across a relatively complex state space. MATLAB has shown to be an efficient tool for solving the motion of the models, and for iterating through the results to identify success or failure of stability criteria. The comparison of speeds in the two approaches clearly shows that the approach of completely analyzing the motion and then parsing the results is much more efficient than calling ODE45 and solving the EOMs for each individual step. The outputs give a valuable insight into the torques ranges that can result achieved stability criteria, and the way that the programs are written will allow for easy expansion – such as introducing extra iteration across additional state spaces (angular position, angular velocity, etc.). One potential challenge that I foresee with the approach that I have taken for the solution is accounting for torque that ramps over time, instead of static torques. This will require further analysis of the inputs to the ODE45 solver and may involve more specific optimizations.

The immediate next step for this work will be to implement functionality that can determine the ZIP behavior of the configurations that this code identifies. In parallel to that development, I want to determine the effective edges of the search space of the parameters that our lab is interested in (the values that make sense for human quiet standing) and implement a mass search utilizing the power of Euler to step through the entire space with a high level of granularity, therefore providing a comprehensive data set that we can begin to work with as we move forward with the ZIP analysis. This will require discussion with my lab and with Euler administration to secure the resources and proper approach.

## 8. Appendix – Equations of Motion

$$\begin{aligned} & j_1 \ddot{\phi}_1 + j_2 \ddot{\phi}_1 + m_2 \ddot{\phi}_2 [l_1^2 + 2l_1 l_{c2} \cos \phi_2] + j_2 \ddot{\phi}_2 + m_2 \ddot{\phi}_2 l_1 l_{c2} \cos \phi_2 + \\ & m_2 l_1 l_{c2} \sin \phi_2 [-2\dot{\phi}_2 \dot{\phi}_1 - \dot{\phi}_2^2] - g[m_1 l_{c1} \sin \phi_1 + m_1 (l_1 \sin \phi_1 + l_{c2} \sin(\phi_1 + \phi_2))] = \tau_1 \end{aligned} \quad [1]$$

$$j_2 \ddot{\phi}_1 + m_2 \ddot{\phi}_1 l_1 l_{c2} \cos \phi_2 + j_2 \ddot{\phi}_2 + m_2 l_1 l_{c2} \dot{\phi}_1^2 \sin \phi_2 - m g l_{c2} \sin(\phi_1 + \phi_2) = \tau_2 \quad [2]$$

## References

[1] Boehm WL, Nichols KM, Gruben KG. Frequency-dependent contributions of sagittal-plane foot force to upright human standing. *J Biomech*. 2019;83:305–9.

(<https://www.sciencedirect.com/science/article/pii/S0021929018308753>)

[2] Shiozawa, K., Lee, J., Russo, M. *et al*. Frequency-dependent force direction elucidates neural control of balance. *J NeuroEngineering Rehabil* **18**, 145 (2021). <https://doi.org/10.1186/s12984-021-00907-2>