

# Computation Time Comparison Between Matlab and C++ Using Launch Windows

Tyler Andrews<sup>1</sup>

*California Polytechnic State University San Luis Obispo, SLO, CA, 93407, USA*

Processing speed between Matlab and C++ was compared by examining launch windows and handling large amounts of data found in pork chop plots. A compilation of code was generated in Matlab to produce the plots and an identical file was created in C++ that was then compiled and run in Matlab to plot the data. This file is known as a MEX-file. This report outlines some of the basics when working with MEX-files and the problems that face users. For Lambert's solver, multi revolution cases were considered and some pork chop plots of single revolution trajectories were plotted. Three different date grids were plotted with different dimensions to determine the difference in processing speed of the two languages.

## Nomenclature

$r$	= radius	[km]
$t$	= time	[sec]
$v$	= velocity	[V]
$\theta$	= true anomaly	[°]
$\chi$	= universal variable	[]

### Subscripts

1	= initial	[]
2	= final	[]
<i>low</i>	= lower bound	[]
<i>up</i>	= upper bound	[]

## I. Introduction

For many orbital mechanics calculations, computer processing must be utilized in order to speed up the calculation process. Matlab is a common tool for orbital calculations because it combines the ability to calculate complex equations and iteration processes with its graphical capabilities, which allows users to visualize the orbit being calculated. Though Matlab is a powerful tool in visualization and calculation of orbital mechanics, it does restrict users in the ability to improve its processing speed. Other programming languages, such as C++, offer a more versatile environment for users to improve the performance of their code. Matlab offers a unique tool to integrate C++ code and utilize Matlab's workspace and graphical capabilities. This tool is known as a MEX-file and allows the user to program in C++ right in the Matlab user interface.

Using C++ instead of Matlab for small programs and functions would increase the speed of the calculations by small amounts. The increase in computing speed of C++ can really be seen when computing large amounts of data and complex programs with large amounts of iteration loops.

Pork chop plots are plots that examine launch windows between two planets. On each of the axes there are an array of dates, one corresponding to the date of departure from the departure planet and one corresponding to the date of arrival to the arrival planet. At each combination of arrival and departure dates, the dates are run through a Lambert's solver method where initial and final spacecraft velocities are found through the use of an iteration method, in this case a universal variables method. Each point of independent arrival and departure combinations has

---

<sup>1</sup> Undergraduate Student, Aerospace Engineering, 1 Grand Ave

a unique set of numbers including a  $c_3$  value leaving the departure planet, an incoming velocity at the sphere of influence at the arrival planet, and a time of flight. This set of values is then plot on a single pork chop plot using a contour plot so that an optimal launch date can be selected.

## II. Procedure

In order to implement C++ in Matlab, a special script file was utilized, called a MEX-file. MEX-files are a special file format designed to take variables from the Matlab workspace and run them through precompiled C++ code located in the MEX-file and process the data. With all MEX-files a header file must be included, `mex.h`, in order to use all functions associated with MEX-files. A MEX-file is similar to regular C++ files, in that it has a main function that Matlab can recognize when calling the function. This main function is called from the following line of code. All user-written MEX-files must be called in this way; there can be no variations in the main function.

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
```

The inputs to this function are important in the execution of the MEX-file. The variable `int nlhs` is an integer that accounts for the number of outputs that the function will return. The variable `mxArray *plhs[]` is an array consisting of all pointers to the data that will be returned by the function. This array has a size of `nlhs` and can be modified anywhere in the code to output any number of variables that is desired. The variable `int nrhs` is an integer corresponding to the number of inputs that the function will take in. The variable `const mxArray *prhs[]` pertains to the inputs to the MEX function. This array cannot be modified throughout the file, the data from the variables can only be read into the file.

In order to read and modify the data inside the MEX-file, a special function is used to read the data from the `mxArray`. This can be accomplished in the following manner:

```
double *p1l;  
p1l = mxGetPr(prhs[0]);
```

All pointers, both to the inputs and outputs, of MEX-functions are pointers to doubles. In order to create an output variable, an extra step must be taken to first create the output matrix. This extra step can be seen below.

```
double *p0l;  
plhs[0] = mxCreateDoubleMatrix(3, 1, mxREAL);  
p0l = mxGetPr(plhs[0]);
```

This creates a  $3 \times 1$  matrix for the output variable and the pointer `p0l` points to that variable, which can be populated by data calculated in the MEX-file.

In order to setup Matlab to be able to recognize and compile MEX-files, Matlab must have a recognizable C++ compiler. To view and select the compiler that Matlab currently recognizes, the following command can be typed into the command line.

```
mex -setup
```

This will allow the user to select a compiler from a list. When working with C++ MEX-files, a C++ compiler must be selected. After the correct compiler is selected, the MEX-file can be compiled by typing the following into the command line:

```
mex mexFile.cpp
```

This compiles the MEX-file and creates a new file with a specific extension based on the operating system that Matlab runs on. For a 32-bit Windows operating system, typing the previous line into the command line will produce a file, `mexFile.w32`; for a 64-bit Windows operating system, the line of code will produce a file titled, `mexFile.w64`, and for Mac operating systems it will produce a file, `mexfile.mexmaci`. It is recommended that each MEX-file is recompiled before running any code on each system, in order to insure that Matlab has the compiler required to run the code. Before a MEX-file can be called from Matlab, it must first be compiled and a file with an extension matching the ones previously mentioned must be in the current folder to run.

There are some problems associated with MEX-files as well. First, Matlab does not have a way to step through and debug each line of code of a MEX-file like it can with `.m` files. To debug the code it must be modified and taken

into another workspace designed to compile and debug C++ code. Second, Matlab cannot terminate a MEX-file as it can with .m files, so the user must be careful as to not run into infinite loops, or Matlab must be quit entirely.

### III. Analysis

In order to calculate the data required to create a pork-chop plot, a Lambert's problem solver method must be utilized. The basis behind a Lambert's problem is that the initial and final radii of the spacecraft and the transit time between the two radii are known and the initial and final velocities are to be calculated. To solve this problem, the selected method was a method that utilized universal variables taken from Vallado's *Fundamentals of Astrodynamics and Applications*.<sup>1</sup>

The first step towards solving Lambert's problem is to calculate the difference in the true anomaly between the two radii.

$$\cos \Delta\theta = \frac{\vec{r}_1 \cdot \vec{r}_2}{r_1 r_2} \quad (1)$$

where  $\vec{r}_1$  and  $\vec{r}_2$  are the two radius vectors. A parameter  $A$  is calculated to determine whether the solution can be calculated. If the value of  $A$  is 0, the orbit cannot be determined using this universal variables method.

$$A = t_m \sqrt{r_1 r_2 (1 + \cos \Delta\theta)} \quad (2)$$

where  $t_m$  is a constant multiplied to  $A$  to account for different solutions to Lambert's problem. For a short-way solution,  $t_m$  is set to 1, and for the long-way solution,  $t_m$  is set to -1.  $t_m$  can also be determined analytically based off of the two radius vectors. If the z-component of the cross product of  $\vec{r}_1$  and  $\vec{r}_2$  is greater than or equal to 0, then  $t_m$  is 1, and if the z-component of the cross product is negative, then  $t_m$  is -1. Once it is determined that the orbit can be calculated, the method can begin the iteration process. For the first iteration, the variable  $\psi_n$ , used to calculate the orbit, is set to 0, where

$$\psi_{low} = -4\pi \quad \psi_{up} = 4\pi^2$$

$\psi_{low}$  and  $\psi_{up}$  are used as initial conditions in the iteration process and specify the range for the variable  $\psi_n$ . For multi-revolution solutions to Lambert's problem,  $\psi_n$  is set to the midpoint between  $\psi_{low}$  and  $\psi_{up}$ , which are defined below with the initial  $\psi_n$  being the midpoint between them.

$$\psi_{low} = 4N^2\pi^2 \quad \psi_{up} = 4(N+1)^2\pi^2$$

After the initial conditions for  $\psi$  are set up the iteration process can be started. First, the value of  $y_n$  must be calculated.

$$y_n = r_1 + r_2 + \frac{A(\psi_n c_3 - 1)}{\sqrt{c_2}} \quad (3)$$

$c_2$  and  $c_3$  are functions of  $\psi_n$  and are defined in Vallado.<sup>1</sup> If  $A$  is greater than 0 and  $y_n$  is less than 0 then  $\psi_{low}$  must be adjusted so that  $y_n$  is greater than 0. After that the universal variable  $\chi_o$  can be calculated.

$$\chi_o = \sqrt{\frac{y_n}{c_2}} \quad (4)$$

After the universal variable is calculated, the time of flight associated with the universal variable can be calculated with it.

$$\Delta t = \frac{\chi_o^3 c_3 + A\sqrt{y_n}}{\sqrt{\mu}} \quad (5)$$

where  $\mu$  is the gravitation parameter of the primary body. If  $\Delta t$  is less than the transfer time for then  $\psi_{low}$  is set to  $\psi_n$ . If  $\Delta t$  is greater than the transfer time for then  $\psi_{up}$  is set to  $\psi_n$ . If the difference between  $\Delta t$  and the given transfer time is not within tolerance, then the process is repeated starting from Eq. 3, where the new value for  $\psi$  is

$$\psi_{n+1} = \frac{\psi_{up} + \psi_{low}}{2} \quad (6)$$

Once  $\Delta t$  converges to within tolerance, the initial and final velocities can be calculated. The Lagrange coefficients can be calculated using the following equations.

$$f = 1 - \frac{y_n}{r_1} \quad (7)$$

$$g = A \sqrt{\frac{y_n}{\mu}} \quad (8)$$

$$\dot{g} = 1 - \frac{y_n}{r_2} \quad (9)$$

Using these Lagrange coefficients, the initial and final velocity vectors can be calculated using the following equations.

$$\vec{v}_1 = \frac{\vec{r}_2 - f\vec{r}_1}{g} \quad (10)$$

$$\vec{v}_2 = \frac{\dot{g}\vec{r}_2 - \vec{r}_1}{g} \quad (11)$$

This method was coded up first in Matlab and can be found in Appendix A. In order to calculate  $v_{inf}$  upon arrival and the  $c_3$  at the departure planet, ephemeris data for Earth and Mars must be calculated, as these two radius vectors will be used as the initial radius vectors for Lambert's problem. The method used to calculate the ephemeris data was taken from *Orbital Mechanics for Engineering Students*, by Howard D. Curtis and utilizes the Julian date of interest to calculate changes in the orbital elements of the planet to update the planets orbital elements, which then get translated into the planets radius and velocity at a specific time. After the planets radius and velocity is known at a specific time, the  $v_{inf}$  of the spacecraft relative to the planet can be found using the following equation.

$$\vec{v}_{inf} = \vec{v}_{sc} - \vec{v}_{pl} \quad (12)$$

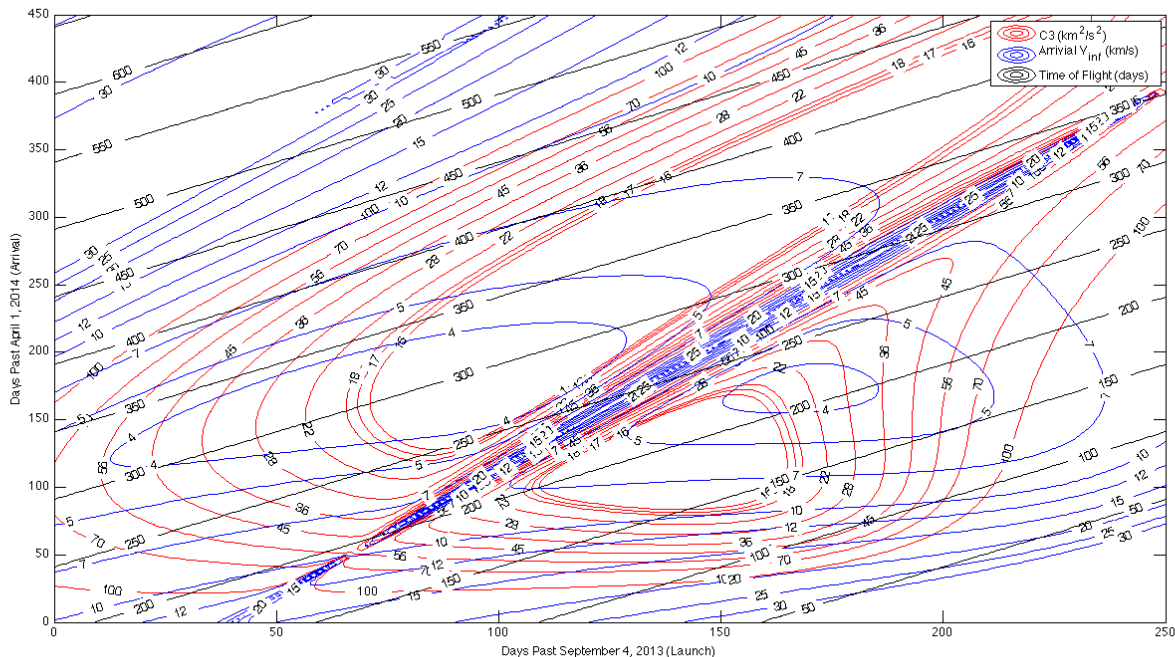
where  $\vec{v}_{sc}$  is the velocity of comes out of the solution to the Lambert's problem, either  $\vec{v}_1$  or  $\vec{v}_2$ , whichever corresponds to the planet it is leaving from or arriving at, and  $\vec{v}_{pl}$  is the velocity of the planet at which the spacecraft is arriving at or leaving from, calculated from the ephemeris data. The  $c_3$  value is calculated by taking the norm of  $\vec{v}_{inf}$  and squaring the result.

#### IV. Results and Discussion

The date range selected for the pork chop plot was the same for both the Matlab code and the C++ code, in order to make the most accurate comparison of time between both sets of code. The initial launch date chosen was September 4, 2013 and array of launch dates continued for 250 days after the initial launch date. The initial arrival date chosen was April 1, 2014 with an array of arrival dates extending to 450 days past the initial arrival date. In Figure 1, the pork chop plot created by both the Matlab code and the C++ code can be seen.

From this figure, the launch window can be determined. For an example, to examine a launch window from the pork chop plot, say a spacecraft must arrive at Mars's sphere of influence with less than a 4 km/s velocity relative to Mars and leave Earth with a  $c_3$  value of less than 16 km<sup>2</sup>/s<sup>2</sup>. Examining first, the type I transfer orbits, which lie below the centerline, the launch window for the spacecraft would be from 150 to 158 days past the initial launch date, from February 1 to February 9, 2014, and it will arrive 160 days after the initial arrival date on September 8,

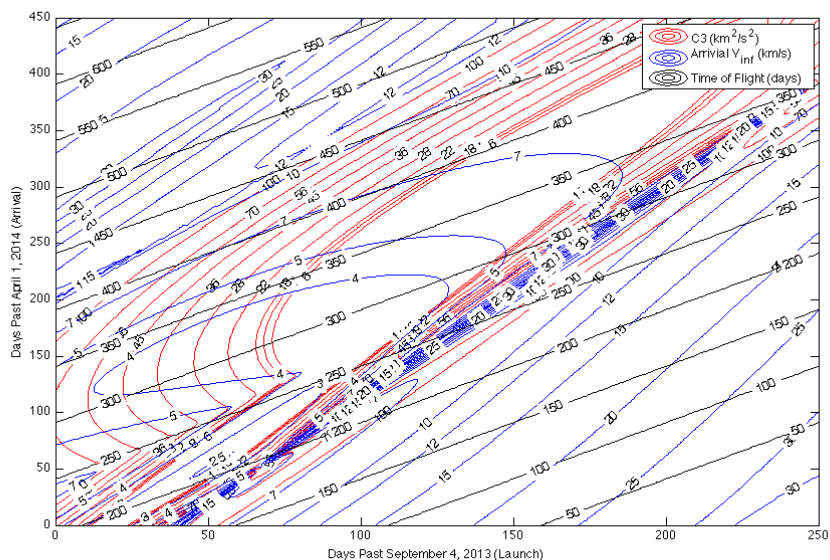
2013. Second, there is another launch window pertaining to the type II orbit transfer. These types of orbits lie above the centerline, and they are defined as an orbit with a difference in true anomaly of greater than  $180^\circ$ . This second launch window is larger than the type I orbit, stretching from 84 to 128 days past the initial launch date from November 27, 2013 to January 10, 2014, and arriving at Mars 210 days past the initial arrival date on October 28, 2013.



**Figure 1.** This figure shows the pork chop plot created by both the Matlab and C++ code.

This code can also produce multi-revolution lamberts plots. The pork chop plot for a single revolution solution can be seen in Figure 2. This plot displays slightly different characteristics than the zero revolution plot from Figure 1. The same arrays of launch and arrival dates were used as the plot in Figure 1. This plot shows the only viable transit option is to travel in a type II trajectory. There are similarities and differences between the single revolution and the zero revolution plots. Much of the type II trajectories look like pork chops, except for the extension in the lower left portion of the plot.

To compare the times between Matlab and C++, three different pork chop plots were calculated: a 100 x 100 plot, a 150 x 150 plot, and a 200 x 200 plot. The duration for the total calculation time for each plot can be found in Table 1. By examining this plot, it is clear that the computation time of C++ is far superior to the processing speed of Matlab. When increasing the dimensions on the plot, it only increases the speed of C++ compared to Matlab.



**Figure 2.** This figure shows the pork chop plot for a single revolution solution.

**Table 1. This table shows the comparison in processing speed between Matlab and C++ for different plot dimensions.**

<b>Plot Dimensions</b>	<b>Matlab Time (sec)</b>	<b>C++ Time (sec)</b>	<b>Factor of Increase in Speed</b>
100 x 100	133.00	0.2706	491.50
150 x 150	305.02	0.5580	546.63
200 x 200	513.67	0.9669	531.25

C++ averages a processing speed that is over 500 times faster than Matlab code. Not only does this apply for this code, but this can also be applied for any other code comparison between Matlab and C++ MEX-files.

## **V. Conclusion**

In comparison, the benefits of speed offered by C++ far outweigh the simplicity of Matlab. Utilizing MEX-files to benefit from the processing speed of C++ with the graphical capabilities of Matlab. Processing speed is especially important when dealing with orbital calculations, since many calculations involve immense optimization with complex equations and algorithms or calculations with a large number of iterations. As the amount of data increases, the computation time for Matlab code increases significantly, therefore Matlab code becomes infeasible for those calculations. C++ is the way to go for orbital calculations because of its speed and versatility.

## **VI. Appendix**

### **A. Matlab Code**

Matlab code can be found with the Aerospace Engineering Department.

## **VII. References**

<sup>1</sup> Vallado, David A. *Fundamentals of Astrodynamics and Applications, Second Edition*. Microcosm Press, Kluwer Academic Publishers. El Segundo, CA. 2001.

<sup>2</sup> Curtis, Howard D. *Orbital Mechanics for Engineering Students, Second Edition*. Butterworth-Heinemann. Great Britain. 2009.