A Simple Leapfrog Integration Scheme to Find Optimal Interplanetary Trajectories

Scott M. Kelley

Thesis Advisor: Dr. Tom Giebultowicz

Oregon State University Department of Physics

March 8, 2018

Table of Contents

Table of Figures

Table of Tables

Abstract

The calculation of interplanetary trajectories is a numeric problem which requires a high degree of precision for the results to be accurate. A computer program was written for this project which uses leapfrog integration combined with Newton's method of iterative root finding to find ideal interplanetary trajectories. Reasonable initial conditions are found by assuming a Hohmann transfer between two orbits. The program accounts for the gravitational influence of all planets in the solar system and seeks a solution which favors the least amount of energy for the transfer. Newton's method of iterative root finding converges exponentially, and leapfrog integration allows for large timesteps which causes the program to find solutions quickly and efficiently.

The program can be used to calculate the ideal trajectory between any two planets in the solar system (including our moon). Additionally, a user can input the absolute initial position and velocity of a craft to model more complicated orbits or trajectories which do not originate on a planet.

1 Introduction

1.1 N-Body Simulations in Computational Astronomy

Several technical challenges which must be overcome for space flight to be possible, including accurate prediction of the trajectory of a spacecraft. On an interplanetary trajectory, a spacecraft spends the majority of its journey mainly under the gravitational influence of the Sun. This can be modeled as a two-body problem which is solvable analytically, but only as a first-order approximation of the craft's true trajectory. Better predictions necessitate the use of numeric integration techniques which consider perturbations caused by gravitational attraction to other celestial bodies.

For this project, a computer program was written which uses a low-error leapfrog integration scheme that is simple to understand and runs very quickly. The program is named SLIIT (**S**imple **L**eapfrog **I**ntegration for **I**nterplanetary **T**rajectories). This project uses SLIIT to model an Earth to Mars transfer, but the methods described can be used to calculate the trajectory between any two planets or the free motion of an object anywhere in our solar system.

1.2 Leapfrog Integration

SLIIT uses leapfrog integration to calculate the trajectory of a spacecraft. This method is chosen because it allows larger timesteps, thus shortening computation time without loss of accuracy. To demonstrate, first consider the estimation of the derivative of an arbitrary function at a given point. The traditional approach, derived directly from the limit definition of the derivative, uses the current value and a future value of the function to calculate the slope of a tangent line.

$$f'(t) \approx \frac{f(t+\Delta t) - f(t)}{\Delta t} \qquad (1)$$

In contrast, the leapfrog method estimates the derivative using values of the function at half-steps in either direction of the point of interest,

$$f'(t) \approx \frac{f\left(t+\frac{\Delta t}{2}\right) - f\left(t-\frac{\Delta t}{2}\right)}{\Delta t} \qquad (2)$$

Both methods use the same size timestep, require the same number of computations, and approach the true derivative as $\Delta t \to 0$; but the leapfrog method more accurately estimates the function's derivative for larger timesteps, as shown in Figure 1.1.

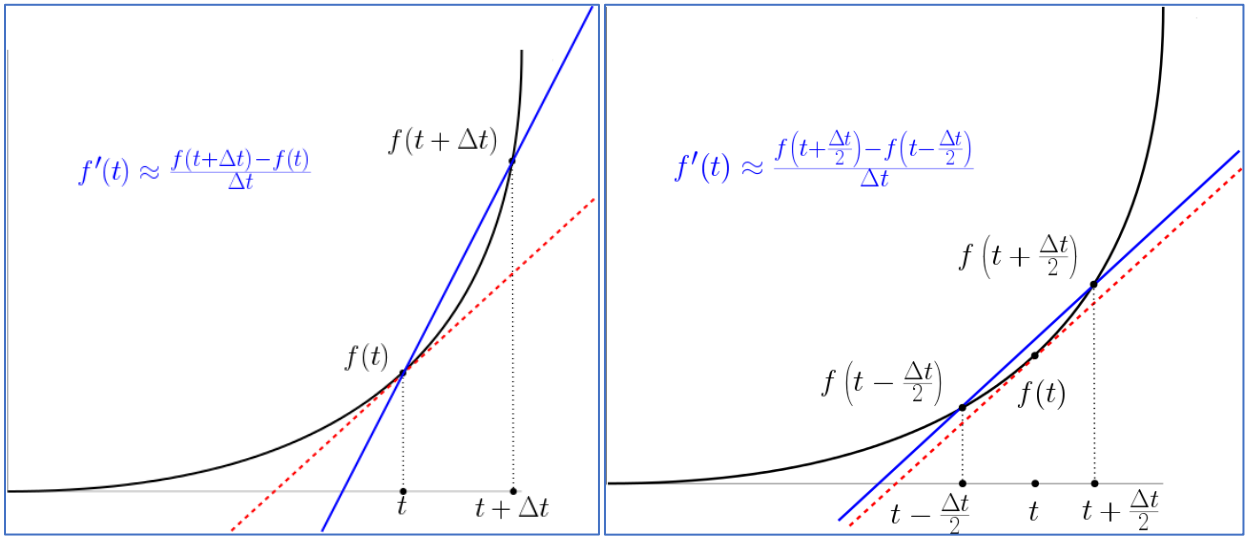

Figure 1.1: Estimation of an arbitrary function's derivative. The red line shows the function's true derivative at time t. The first figure shows the traditional method of estimating its derivative, while the second figure shows the leapfrog method. Both require the same number of computations and both approach the true derivative as $\Delta t \to 0$.

Leapfrog integration similarly uses an integrand's current value to calculate the value of the integral spaced with half-steps around it. Figure 1.2 shows how this is applied to the prediction of the trajectory of a spacecraft. The future position is calculated using the position from the current timestep and the velocity from the future half-timestep. The calculation of the velocity at half-steps allows the simulation to use larger timesteps and leads to better predictions of the craft's trajectory. Additionally, the use of half-steps forces the craft's velocity to be tangential to its orbit, conserving angular momentum and allowing for the simulation be run for longer without compiling error.

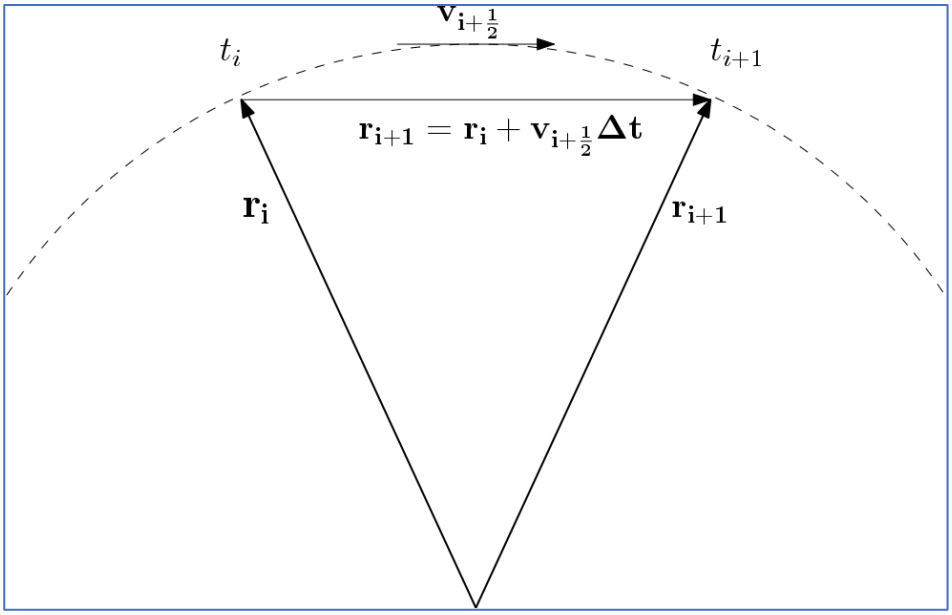

Figure 1.2: Leapfrog integration with half-steps. The future position is calculated using the position of the current timestep and the velocity from the future half-timestep.

The basic equations of motion of the leapfrog integration scheme are

$$\begin{aligned} \boldsymbol{r}_{i+1} &= \boldsymbol{r}_i + \boldsymbol{v}_{i+\frac{1}{2}}\,\Delta t \\ \boldsymbol{v}_{i+\frac{1}{2}} &= \boldsymbol{v}_{i-\frac{1}{2}} + \boldsymbol{a}_i\,\Delta t \end{aligned} \qquad (3)$$

where the acceleration is the sum of the external forces on the body divided by its mass.

$$\boldsymbol{a_i} = \frac{\mathbf{1}}{\boldsymbol{m_i}} \sum \boldsymbol{F_j^{ext}} \tag{4}$$

This system of equations can be rewritten without the explicit use of half-steps by first expressing the velocity term for a full step in two distinct half-steps

$$\begin{aligned} \boldsymbol{v}_{\boldsymbol{i+\frac{1}{2}}} &= \boldsymbol{v_i} + \frac{\boldsymbol{a}_i}{2}\, \Delta t \\ \boldsymbol{v}_{i+1} &= \boldsymbol{v}_{i+\frac{1}{2}} + \frac{\boldsymbol{a}_{i+1}}{2}\, \Delta t \end{aligned} \tag{5}$$

Combining Equation (3) with Equation (5) allows the equations of motion to be written strictly in terms of integer timesteps

$$\begin{aligned} \boldsymbol{v}_{i+1} &= \boldsymbol{v}_i + \frac{(\boldsymbol{a}_i + \boldsymbol{a}_{i+1})}{2}\, \Delta t \\ \boldsymbol{r}_{i+1} &= \boldsymbol{r}_i + \boldsymbol{v}_i\, \Delta t + \frac{\boldsymbol{a}_i}{2}\, \Delta t^2 \end{aligned} \tag{6}$$

This formation makes programming easier without any loss of functionality [1]. Section 2.1 describes how Equation (6) is programed in SLIIT.

1.3 Cubic Interpolation

While it is necessary to consider the gravitational influences of all the planets in the solar system on a spacecraft's trajectory, it is unnecessary to calculate the position of each planet

every time SLIIT is run[1]. Therefore, the planets' positions are tabulated and referenced at runtime. To get better fidelity than the values used in the reference tables, each planet's position is interpolated at each timestep. For instance, the reference tables used for this project list the positions of the planets every 60 minutes, but it may be necessary to inquire their positions several times a second and therefore interpolation of the tables is required. The path of each body is well-behaved enough that a small section of the orbit is estimated as a cubic polynomial of the form

$$P(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \qquad (7)$$

where $P(t)$ is the interpolated position and $a_0, a_1, a_2, \text{and } a_3$ are unknown coefficients of the polynomial.

A property of polynomials is that for N ordered pairs of positions and times (r, t), a unique polynomial of order $N - 1$ exists which exactly passes through each of the points. The method of Lagrange polynomials is used to solve for this unique polynomial and is defined as

$$P(t) = \sum_{i=0}^{N-1} r_i \prod_{j=0, j\neq i}^{N-1} \frac{t - t_j}{t_i - t_j}. \qquad (8)$$

where $r_i, \dots, r_j$ and $t_i, \dots, t_j$ are known positions and times and t is the time for which the interpolated position is desired [2]. Section 2.1 describes how Equation (8) is programed in SLIIT.

---

[1] The addition of a spacecraft in the solar system has negligible impact on the planets' orbits. If SLIIT is used to model the trajectory of a planetary-sized object, then this assumption may not hold.

Lagrange polynomials find the proper function by summing unique polynomials which each evaluate to one of the known positions at a known time and evaluate to zero at all other known times; when t is a known value then $P(t_i) = r_i$. For all other times, $P(t)$ returns a linear combination with contributions from each of the ordered pairs. Figure 1.3 shows an arbitrary data set which represents a slightly-curved line. The solid line in the figure is a single polynomial which passes through all four known points and can reasonably be used to interpolate the data[2].

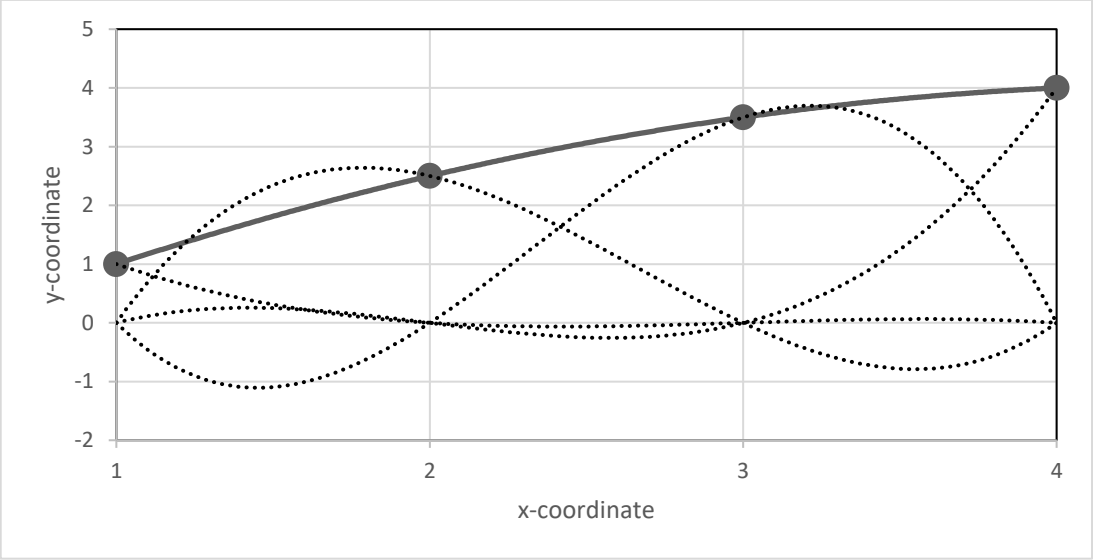

Figure 1.3: Cubic Interpolation using the method of Legrange Polynomials to find a curve through the arbitrary points (1.0, 1.0), (2.0, 2.5), (3.0, 3.5) and (4.0, 4.0). The dotted curves are polynomials which each pass through a single known point and evaluate to zero at the other three known points. The solid curve is the sum of the dotted curves; a single polynomial which passes through all known points.

[2] Care must be taken when using a single polynomial to interpolate a data set. While a polynomial can always be found which passes through every known point, if the data are not well-behaved then the fitting polynomial will be highly oscillatory and not valid for interpolation. This project uses a single polynomial to interpolate a table where the known positions of each planet are provided in one-hour intervals, making for a very slightly curved line.

1.4 Initial Conditions

An iterative root finding algorithm was used to calculate the ideal trajectory from Earth to Mars. There are several trajectories a craft could take and therefore we must have a method for finding a good initial guess for the position and velocity of the spacecraft. The ideal trajectory is the one which needs the least amount of energy to transfer from Earth's orbit to that of Mars. To find an approximation of the ideal trajectory a few simplifying assumptions are made; the orbits of Earth and Mars are assumed to be coplanar circles, the craft is assumed to follow an elliptical path about the Sun where attraction to all other planets is neglected, and the craft is assumed to be put into orbit with an instantaneous impulse. With these assumptions, the trajectory can be modeled as a Hohmann transfer.

The Hohmann transfer [3] models the trajectory as an elliptical orbit which has its perigee at Earth's orbit and apogee at Mars' orbit (as can be seen in Figure 1.4). There is no expectation for the Hohmann trajectory to be correct, but it gives reasonable initial conditions which can be iterated to find a more precise solution.

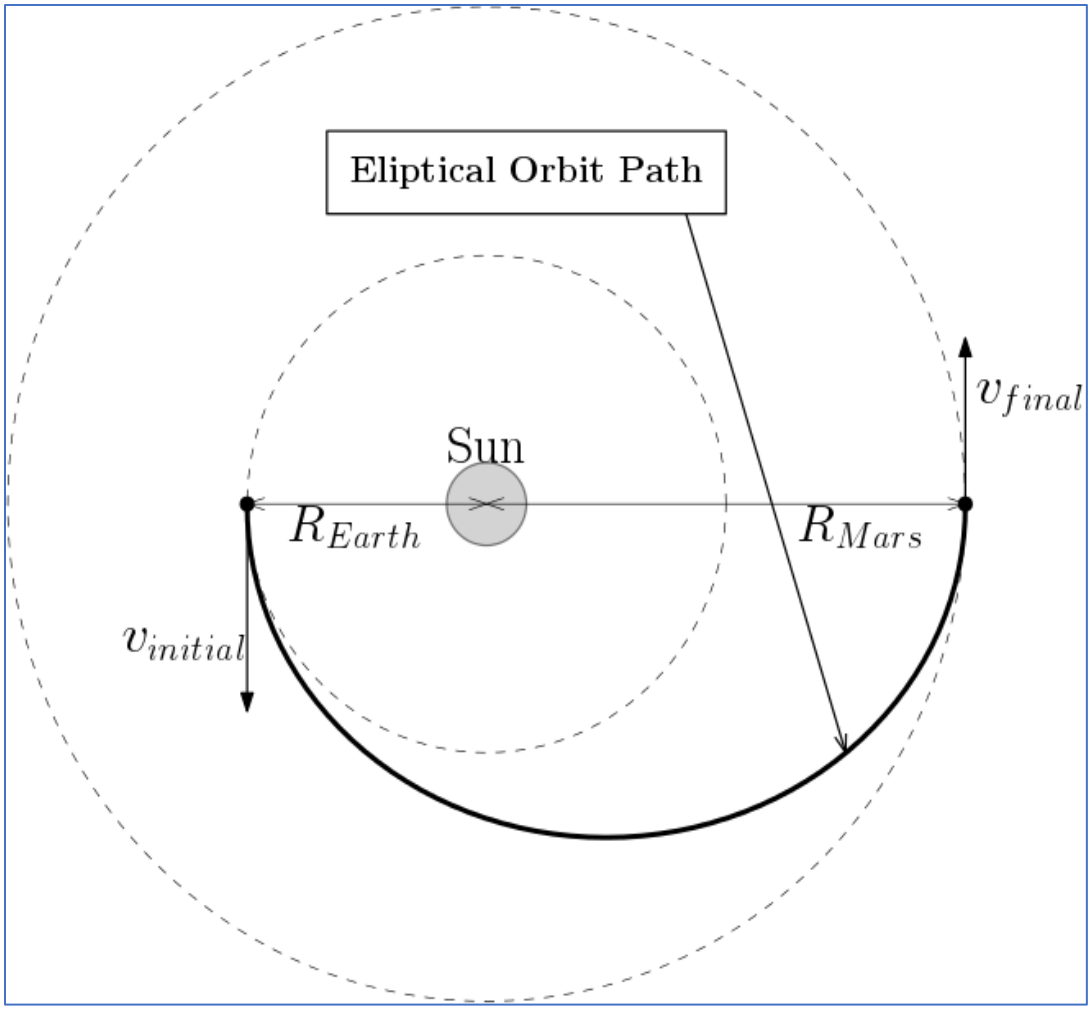

Figure 1.4: Hohmann Transfer from Earth to Mars. The orbits of the planets are modelled as co-planar, circular orbits at each planet's mean distance from the sun. The spacecraft follows an elliptical trajectory which has its perigee at Earth's orbit and apogee at Mars' orbit.

1.4.1 Planetary Sphere of Influence

As mentioned before, most of the time a craft spends in interplanetary flight is under the sole influence of the sun. For this project, instantaneous impulse to start the craft on its trajectory was assumed. In reality, the change of velocity into the elliptical orbit would take some time, but an instantaneous impulse is a reasonable assumption given that the burn time of the craft

to leave Earth's sphere of influence is much smaller than the period of the elliptical orbit of the transfer.

The Earth's sphere of influence is defined as the distance from Earth where the gravitational pull of the Sun is greater than that of Earth. This distance is calculated by equating the potential energy between a craft and Earth with the potential energy between the craft and the Sun.

$$E = -G\frac{m_{craft}M_{Sun}}{r_{craft\rightarrow Sun}} = -G\frac{m_{craft}M_{Earth}}{r_{craft\rightarrow Earth}} \quad (9)$$

$$r_{craft\rightarrow Earth} = \frac{M_{Earth}}{M_{Sun}} r_{craft\rightarrow Sun} \quad (10)$$

In the case of a craft leaving earth's orbit, $r_{craft\rightarrow Sun}$ is about the same as Earth's orbital radius and $r_{craft\rightarrow Earth}$ is the Earth's sphere of influence, SOI_{Earth}. The Earth's orbital radius is 1 AU, so when working in astronomical units SOI_{Earth} reduces to the ratio of Earth's and the Sun's masses.

$$SOI_{Earth} = \frac{M_{Earth}}{M_{Sun}}\ (AU) \approx 3.0035 * 10^{-6}\ (AU) \quad (11)$$

1.4.2 Hohmann Transfer – Initial Velocity of Craft

By closer examination of Figure 1.4 we see that the initial velocity and final velocity of the craft are co-linear. This is needed in order to minimize the energy of the transfer. For proof of this, consider the magnitudes of the initial velocity, final velocity, and the change in velocity vectors for a craft with an arbitrary trajectory (Figure 1.5).

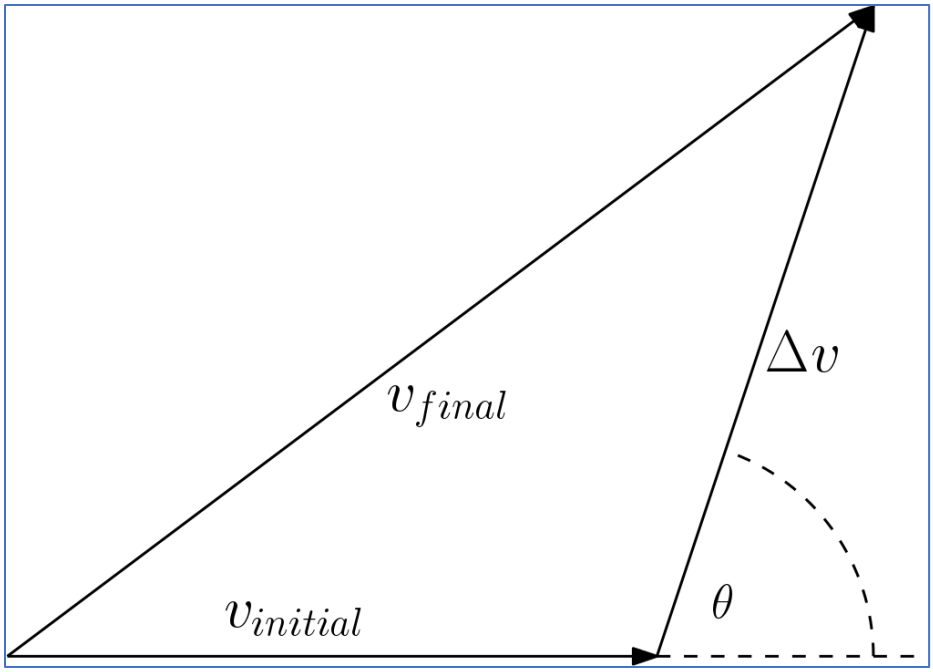

Figure 1.5: Generalized change in velocity for an arbitrary trajectory.

By the law of cosines, the final velocity is expressed as

$$v_{final}^2 = v_{initial}^2 + \Delta v^2 + 2v_{initial}\,\Delta v\,\cos\theta \qquad (12)$$

and the energy change associated with this velocity change is

$$\Delta E = \frac{1}{2}\Delta v^2 + V_{initial}\Delta v\,\cos\theta. \qquad (13)$$

For a transfer from two circular orbits we know Δv regardless of the path since the craft must start at the velocity of the initial orbit and end at the velocity of the final orbit. Equation (13) shows that for any given Δv the change in energy will be largest when $\theta = 0$. In short, we get the most "bang for our buck" from an energy gained-per-impulse-required perspective when

the initial and final velocities of the craft are co-linear. In the case of an elliptical trajectory this is where the inner and outer orbits are at the perigee and apogee of the orbit, respectively. By conservation of energy the velocity at the perigee of the transfer orbit from Earth to Mars, v_π, is

$$v_\pi^2 = 2GM_{Sun}\left(\frac{1}{R_{Earth}} - \frac{1}{R_{Earth} + R_{Mars}}\right) \quad (14)$$

To calculate the impulse needed to move the craft from Earth's orbit into the elliptical trajectory we subtract the velocity of the circular orbit, v_{Earth}, from v_π. Again, by conservation of energy we calculate that

$$v_{Earth} = \sqrt{\frac{GM_{Sun}}{R_{Earth}}} \quad (15)$$

and therefore, the change in velocity is

$$\Delta v_\pi = v_\pi - v_{Earth} = \sqrt{\frac{GM_{Sun}}{R_{Earth}}}\left(\sqrt{\frac{2R_{Mars}}{R_{Earth} + R_{Mars}}} - 1\right). \quad (16)$$

The same method can be used to calculate the velocities at Mars' orbit, but for this project the focus is on the initial conditions necessary to start the proper elliptical trajectory. The maneuvers necessary after the craft is within Mars' sphere of influence are neglected.

1.4.3 Hohmann Transfer – Determining the Ideal Launch Date

We have found the elliptical path the craft must take, but now must consider the phase angle between Earth and Mars at the time of launch to ensure that the spacecraft reaches Mars' orbit at the same time the planet does. The craft is traveling half the period of an ellipse; by Kepler's third law the period of an ellipse of a planet orbiting the Sun is

$$T = 2\pi\sqrt{\frac{a^3}{GM_{Sun}}} \tag{17}$$

where a is the length of the semi-major axis. By taking another look at Figure 1.4 we see that the transfer orbit has a semi-major axis of

$$a = \frac{R_{Earth} + R_{Mars}}{2}. \tag{18}$$

By equations (17) and (18) the time of flight of the craft is calculated to be

$$T_{flight} = \pi\sqrt{\frac{(R_{Earth}+R_{Mars})^3}{8GM_{Sun}}}. \tag{19}$$

To find the desired phase angle between Earth and Mars at launch, we first calculate the angular distance that Mars travels in T_{flight}, as shown in Figure 1.6.

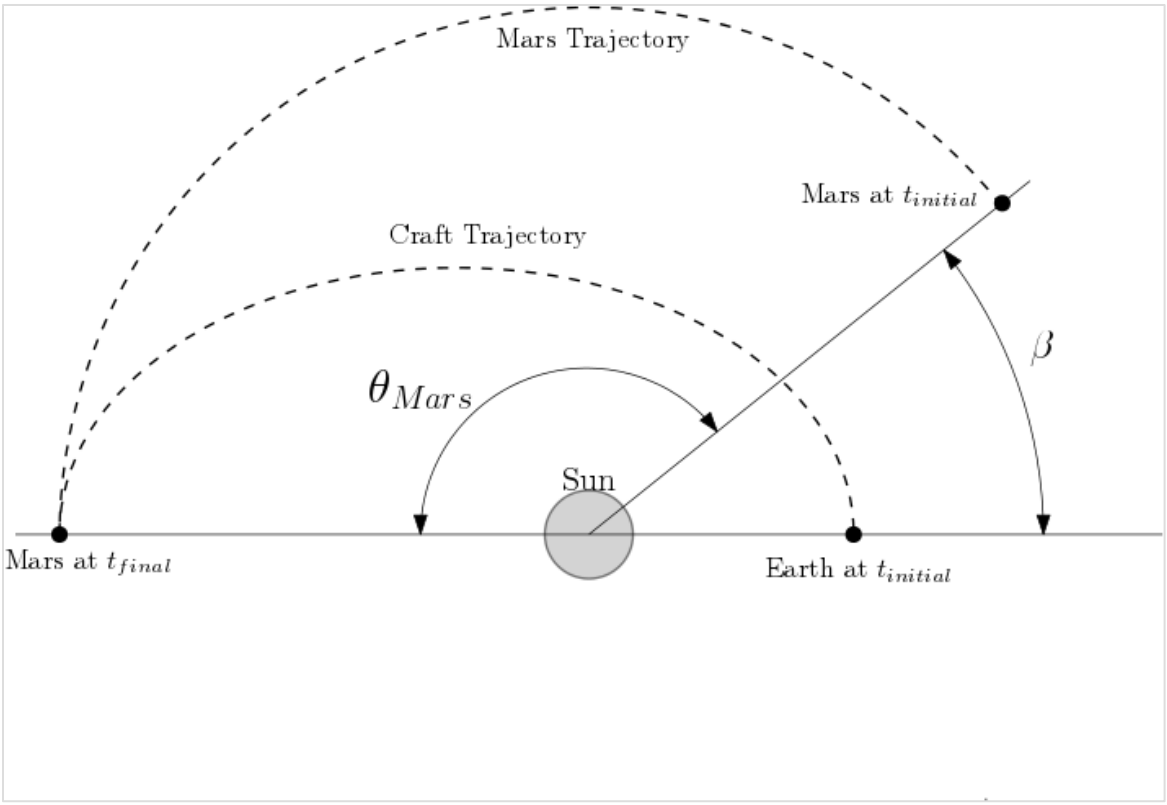

Figure 1.6: Relative angle between Earth and Mars at launch. The spacecraft must arrive at Mars' orbit at the same time and location as the planet itself. The period of the craft's trajectory is known, and Mars' angular velocity is known which allows for the calculation of ideal angle between Earth and Mars at the time of launch (β).

The angular displacement, θ_{Mars}, is found by multiplying its angular velocity, ω_{Mars}, by T_{flight}, where

$$\omega_{Mars} = \sqrt{\frac{GM_{Sun}}{R_{Mars}^3}} \tag{20}$$

and

$$\theta_{Mars} = \omega_{Mars} T_{flight} = \frac{\pi}{2\sqrt{2}} \sqrt{\left(\frac{R_{Earth}}{R_{Mars}} + 1\right)^3}. \tag{21}$$

As discussed in Section 1.4.2, the initial and final velocity vectors of the craft are co-linear. Therefore, the craft's trajectory covers π radians and the ideal phase angle between Earth and Mars at launch is

$$\beta_{ideal} = \pi - \theta_{Mars}$$
$$= \pi\left(1 - \frac{1}{2\sqrt{2}}\sqrt{\left(\frac{R_{Earth}}{R_{Mars}} + 1\right)^3}\right) \qquad (22)$$
$$\approx 0.245\,\pi\,.$$

1.5 Iterative Root Finding

After using the assumptions discussed above to give a reasonable guess of the initial velocity of the craft, it is necessary to fine-tune the velocity to find a solution which considers perturbations due to the actual path of the destination planet and the gravitational forces from other planets in the solar system. Newton's Method iteratively finds a root by calculating the value of a function, $f(x)$, and its derivative, $f'(x)$, at an approximation to a zero of $f(x_n)$. Then the function is replaced by its tangent line approximation at x_n, which is then solved for the x-intercept, as seen in equation (23). This value is taken as the next approximation, x_{n+1}, as can be seen in Figure 1.7.

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \qquad (23)$$

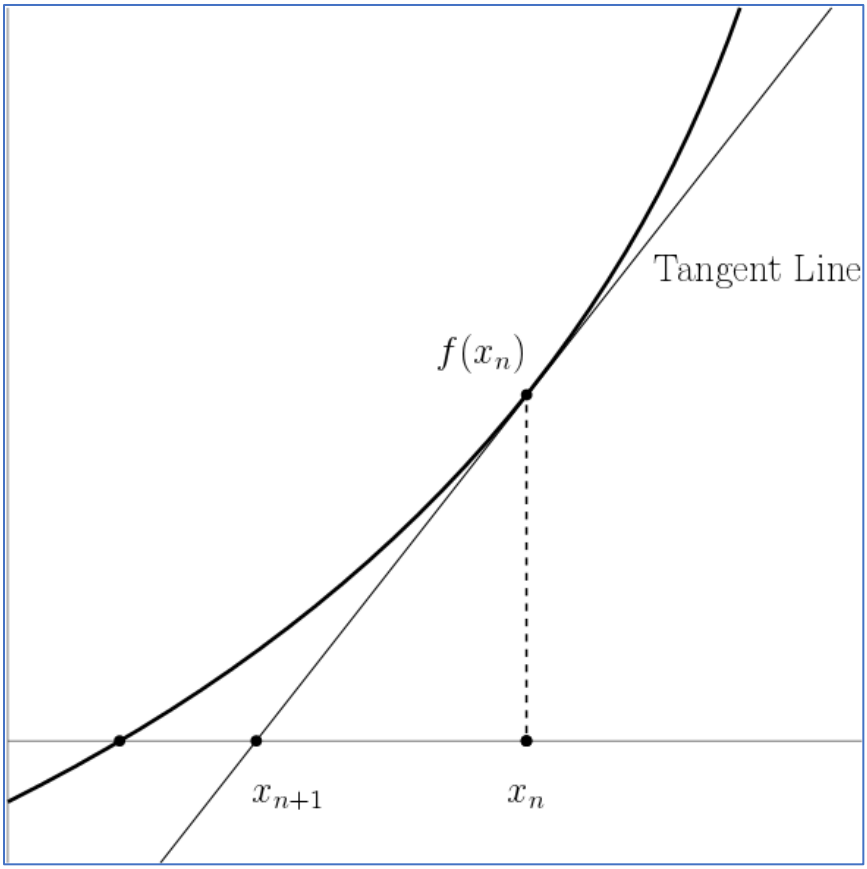

Figure 1.7: Newton's method for iteratively finding the root of a function. At an approximate root, the function's derivative is linearized and evaluated at its x-intercept to provide a better approximation of the function's root. The process is repeated until the root is found to the desired precision.

The explicit expression for x_{n+1} is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{24}$$

Newton's method is generalized to solve a multivariable system of N-equations and M-unknowns

$$\begin{cases} f_1(x_1, \cdots, x_M) = f_1(\boldsymbol{x}) = 0 \\ f_2(x_1, \cdots, x_M) = f_2(\boldsymbol{x}) = 0 \\ \vdots \\ f_N(x_1, \cdots, x_M) = f_N(\boldsymbol{x}) = 0 \end{cases} \tag{25}$$

where $\boldsymbol{x}$ is defined as a vector of independent variables

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}. \tag{26}$$

The system is further simplified as a vector function

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_N(\boldsymbol{x}) \end{bmatrix} \tag{27}$$

and the system in equation (25) is expressed as

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}. \tag{28}$$

Like in the 1-D case, when given an approximation of a root $\boldsymbol{x}_n$ the system is linearized to find the next approximation $\boldsymbol{x}_{n+1}$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \boldsymbol{J}_n^{-1}\boldsymbol{f}(\boldsymbol{x}_n) \tag{29}$$

where $\boldsymbol{J}_n$ is the Jacobian matrix, defined as

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{d}{d\boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \cdots & \frac{\partial f_N}{\partial x_M} \end{bmatrix}. \tag{30}$$

When there are as many equations as unknowns (N=M) the Jacobian is square and the inverse $\boldsymbol{J}_n^{-1}$ is calculated according to the usual linear algebra rules. However, in the case of an over constrained system (N > M) the Jacobian is not square and $\boldsymbol{J}^{-1}$ is calculated as the pseudo-inverse

$$\boldsymbol{J^{-1}} = \left(\boldsymbol{J^T J}\right)^{\boldsymbol{-1}}\boldsymbol{J^T} \tag{31}$$

2 Methods

With this being a computational experiment, the majority of the methods used are programming adaptations of the techniques discussed in the Introduction section above. Figure 2.1 shows an overview of how the SLIIT program operates. Most of the code deals with data management and visualization and is not reproduced here. This section will go into greater detail of the numeric methods used to implement Lagrange polynomials, leapfrog integration, and Newton's method of iterative root finding. The specifics of the choice of a launch window and craft initial velocity are also discussed.

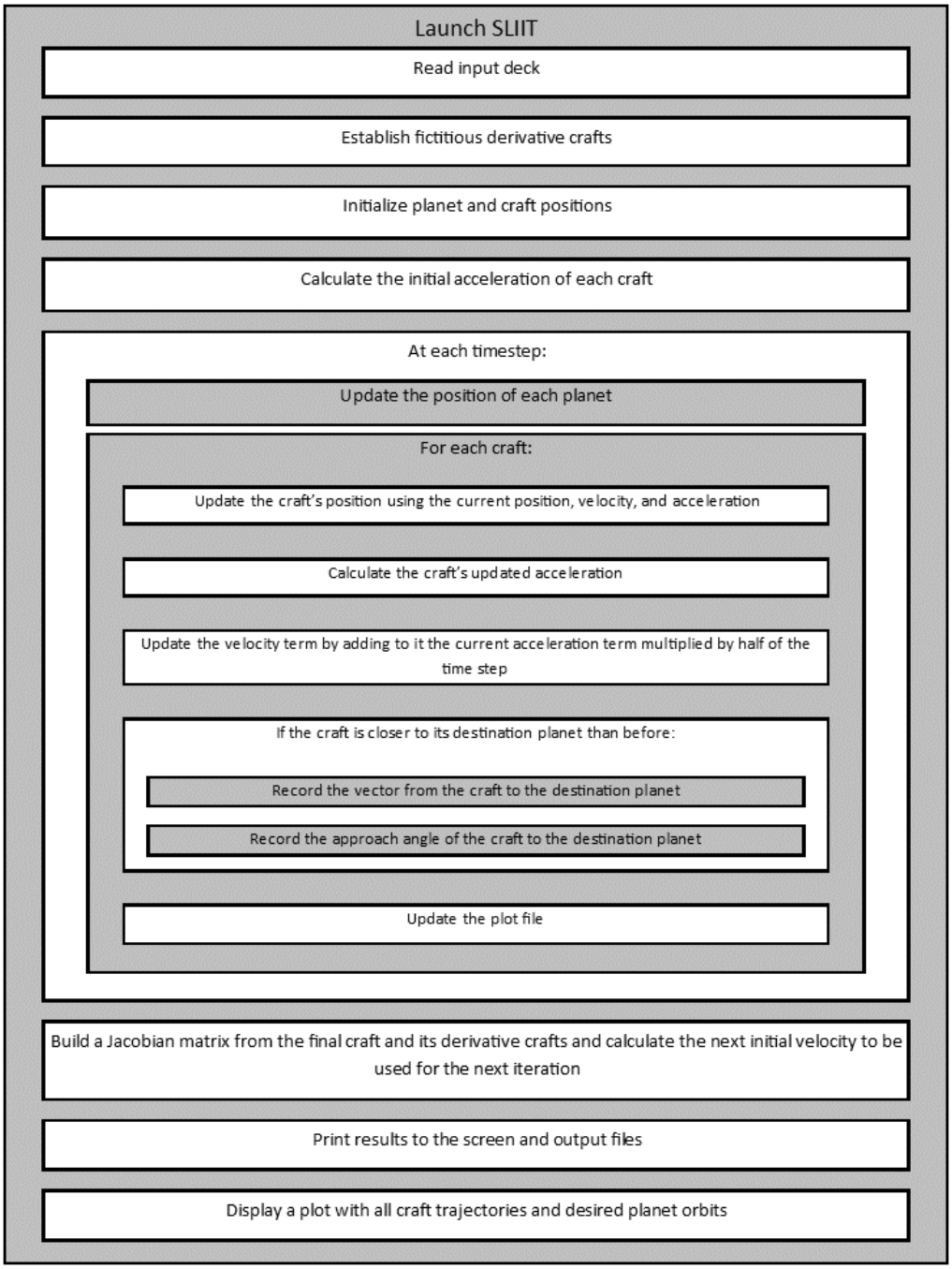

Figure 2.1: Description of SLIIT, the program written for this project. *SLIIT requires a specifically formatted input file and an ephemeris file for each planet, and outputs a summarizing output file and a data file formatted for easy plotting with GNUplot.*

2.1 Programing Cubic Interpolation Scheme

NASA graciously provides a tool online which allows access to JPL's HORIZONS system and can be used to generate ephemerides for several bodies in our solar system [4]. With this tool we are able to download very precise tables with the positions of planets with respect to the solar system's barycenter. Figure 2.2 shows a sample of the ephemeris file holding data for Earth.

```
*******************************************************************************
Ephemeris / WWW_USER Mon Apr 18 09:51:03 2016 Pasadena, USA      / Horizons
*******************************************************************************
Target body name: Earth (399)                    {source: DE431mx}
Center body name: Solar System Barycenter (0)    {source: DE431mx}
Center-site name: BODY CENTER
*******************************************************************************
Start time      : A.D. 2016-Jan-01 00:00:00.0000 TDB
Stop  time      : A.D. 2026-Jan-01 00:00:00.0000 TDB
Step-size       : 60 minutes
*******************************************************************************
Center geodetic : 0.00000000,0.00000000,0.0000000 {E-lon(deg),Lat(deg),Alt(km)}
Center cylindric: 0.00000000,0.00000000,0.0000000 {E-lon(deg),Dxy(km),Dz(km)}
Center radii    : (undefined)
Output units    : AU-D
Output format   : 01
Reference frame : ICRF/J2000.0
Output type     : GEOMETRIC cartesian states
Coordinate systm: Ecliptic and Mean Equinox of Reference Epoch
*******************************************************************************
JDTDB
   X     Y     Z
*******************************************************************************
$$SOE
2457388.500000000 = A.D. 2016-Jan-01 00:00:00.0000 (TDB)
  -1.630229002588497E-01  9.704723344534316E-01 -1.955367328932975E-04
2457388.541666667 = A.D. 2016-Jan-01 01:00:00.0000 (TDB)
  -1.637409328555978E-01  9.703483587900352E-01 -1.955551933805082E-04
2457388.583333333 = A.D. 2016-Jan-01 02:00:00.0000 (TDB)
  -1.644588775901752E-01  9.702238597083108E-01 -1.955736255136229E-04
2457388.625000000 = A.D. 2016-Jan-01 03:00:00.0000 (TDB)
  -1.651767340746068E-01  9.700988372524350E-01 -1.955920272852891E-04
2457388.666666667 = A.D. 2016-Jan-01 04:00:00.0000 (TDB)
  -1.658945019207976E-01  9.699732914669018E-01 -1.956103966913419E-04
2457388.708333333 = A.D. 2016-Jan-01 05:00:00.0000 (TDB)
  -1.666121807405341E-01  9.698472223965223E-01 -1.956287317307539E-04
2457388.750000000 = A.D. 2016-Jan-01 06:00:00.0000 (TDB)
  -1.673297701454850E-01  9.697206300864263E-01 -1.956470304054363E-04
2457388.791666667 = A.D. 2016-Jan-01 07:00:00.0000 (TDB)
  -1.680472697472016E-01  9.695935145820623E-01 -1.956652907206868E-04
2457388.833333333 = A.D. 2016-Jan-01 08:00:00.0000 (TDB)
  -1.687646791571186E-01  9.694658759291983E-01 -1.956835106848412E-04
```

Figure 2.2: Sample Ephimeris for Earth from the JPL Horizons website. The position of the planet is given at one-hour intervals in cartesian coordinates using the ecliptic and mean equinox of reference epoch system with the origin at the solar system barycenter.

Functions were written to navigate the files and interpolate for positions that fall between the values listed. Below is the pseudocode of the interpolation function, which follows Equation (8).

```
CubicInterpolate()
//Use Lagrange polynomial to interpolate a value given 4 known points.
//returns interpolated position at desired time
```

```
    rr = 0.0;
    for (i = 0; i < 4; i++)
        L = 1.0;
        for (j = 0; j < 4; j++)
            if (i != j) L *= (tt - t[j]) / (t[i] - t[j]);
        rr += r[i] * L;
```

Table 1 describes the variables used in CubicInterpolate().

Table 1: Variables used in SLIIT's CubicInterpolate() function

Variable	Description
r	Vector of 4 known positions
t	Vector of 4 known times associated with the positions in r
tt	Time of desired interpolated position
rr	Interpolated position at time tt is returned

2.2 Programming Leapfrog Integration Scheme.

The following pseudocode shows the functions used to implement the leapfrog integration scheme. The first function, CraftAccelerations(), follows Equation (4) to calculate the instantaneous acceleration of the spacecraft.

```
CraftAccelerations()
       //calculates the acceleration of the spacecraft
        for (k = 0; k < 3; k++) craft.A[k] = 0.0; // reset accelerations
        for (j = 0; j < num_bdys; j++)  //for each body...
            for (k = 0; k < 3; k++) //calculate difference between position vectors
                r[k] = craft.R[k] - body.R[k];

            //Calculate r2neg3
            r2neg3 = VectorMag(r);
            r2neg3 = pow(r2neg3, -3);

            //Sum the accelerations in each dimension
            for (k = 0; k < 3; k++)
                craft.A[k] += (-1.0)*G*body[j].mass*r2neg3*r[k];
```

Table 2 describes the variables used in CraftAccelerations().

Table 2: Variables used in SLIIT's CraftAccelerations() function

Variable	Description
G	Newton gravitational constant
r	3-dimensional position vector to hold the difference between the craft and a planet
k	Counter to track 3 spatial dimensions
j	Counter to track each planet
num_bdys	Total number of planets considered
craft.R	Craft position
body.R	Body position
r2neg3	inverse of r-cubed
VectorMag	A function to calculate the scalar magnitude of a vector
body.mass	Mass of body
craft.A	Craft acceleration vector

The following pseudocode shows the function used to update the position of a spacecraft in motion. It is assumed that the craft's acceleration and velocity vectors are populated from the previous timestep. This function follows Equation (6).

```
UpdateCrafts()
        for (k = 0; k < 3; k++) // for each dimension...
               //update position
               craft.R[k] += craft.V[k]*dt + craft.A[k]*(dt*dt)/2;
               //update velocity for current timestep
               craft.V[k] += craft.A[k]*dt/2;

       //Call CraftAccelerations() to calculate accelerations for next timestep
       CraftAccelerations();
       // complete velocity equation in preparation for next iteration
       for (k = 0; k < 3; k++)
               craft.V[k] += craft.A[k]*dt/2;
```

Table 3 describes the variable used in UpdateCrafts().

Table 3: Variables used in SLIIT's UpdateCrafts() function

Variable	Description
k	Counter to track 3 spatial dimensions
craft.R	craft position
craft.V	craft velocity
craft.A	craft acceleration
dt	timestep

2.3 Calculating Initial Launch Date

To choose an initial launch date it was assumed that the craft would perform a Hohmann transfer which needs a specific phase angle between the initial planet and the destination planet at the time of launch. The ephemeris data for Earth and Mars is used to calculate the phase angle between the two planets. The phase angle is calculated by subtracting Earth's position vector from Mars' position vector and taking the arctangent of the resulting vector.

$$\beta = tan^{-1}\left(\frac{y_{Mars} - y_{Earth}}{x_{Mars} - x_{Earth}}\right) \qquad (32)$$

Figure 2.3 shows a plot of the phase angle between Earth and Mars as a function of the date. For this project we looked at the planets positions from the years 2016 through 2025. We see that every angle is covered 5 times in the 10-year period. The discontinuities in the plot are where Earth and Mars are in line and the angle "jumps" from 2π to 0.

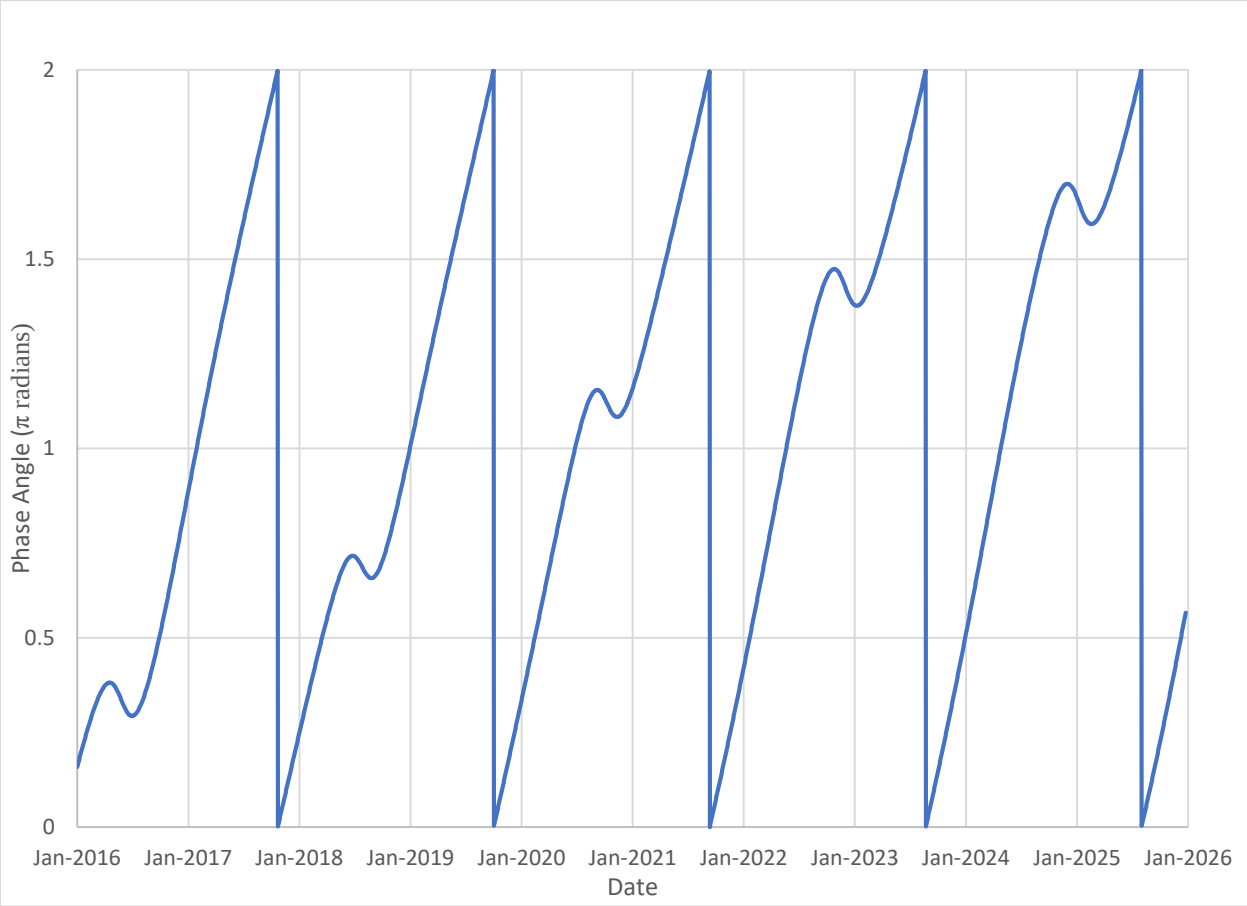

Figure 2.3: Phase angle between Mars and Earth as a function of date. We see that every angle is covered 5 times in the 10-year period shown. The discontinuities in the plot are where Earth and Mars are in line and the angle "jumps" from 2π to 0.

For this project it was chosen to focus on the launch date in the 2017-2018 range. By equation (22) the ideal phase angle at launch is where $\beta_{ideal} \approx 0.25\,\pi$. Figure 2.4 shows the same plot as Figure 2.3 with a closer look those dates and includes a horizontal line at β_{ideal}. The intersection between the phase angle curve and the ideal angle curve is on December 31, 2017; that is, therefore, the launch date chosen.

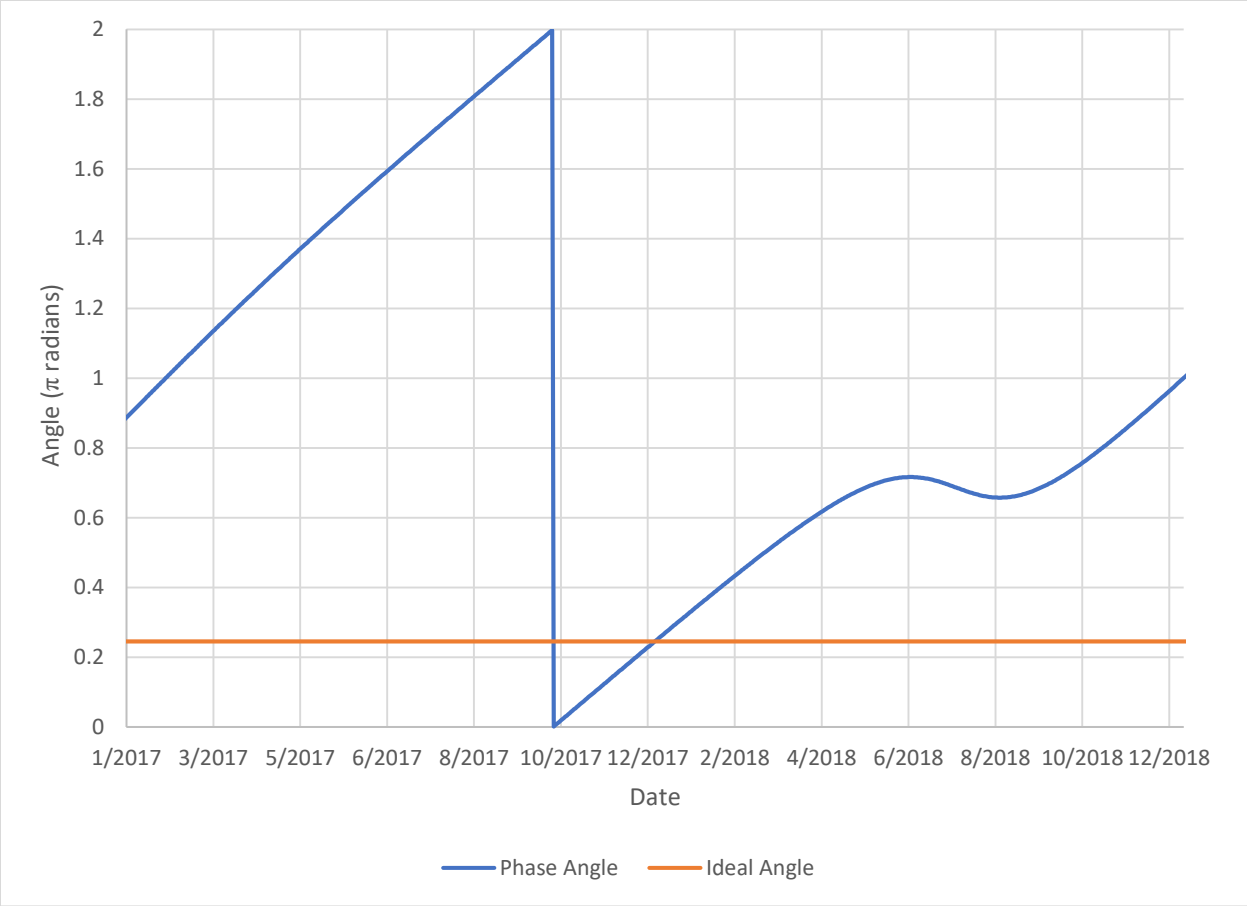

Figure 2.4: Selection of a launch date. The blue curve represents the phase angle between Earth and Mars The orange curve represents the ideal phase angle at launch (0.25π radians). The intersection between the phase angle curve and the ideal angle curve is on December 31, 2017.

2.4 Estimating Initial Craft Velocity

The position of Earth was referenced in the ephemeras files for the launch date and Equation (16) was used to calculate the magnitude of the initial velocity of the craft. To perform a Hohmann transfer the launch trajectory is tangential to the Earth's orbit. For the initial guess, it was assumed that Earth and Mars are coplanar and therefore the z-component of the vector is zero. The other two components of the unit velocity vector are calculated as

$$\hat{v}_{x,0} = \frac{-r_{y,0}}{\sqrt{r_{x,0}^2 + r_{y,0}^2}}$$
$$\hat{v}_{y,0} = \frac{r_{x,0}}{\sqrt{r_{x,0}^2 + r_{y,0}^2}}. \quad (33)$$

With these considerations, the initial velocity of the craft on December 31, 2017 is

$$\boldsymbol{v}_0 = \begin{bmatrix} -1.86 * 10^{-2} \\ -3.15 * 10^{-3} \\ 0.00 \end{bmatrix} \frac{AU}{day}. \quad (34)$$

To place the craft outside of Earth's sphere of influence at launch the initial position of the craft was offset in the direction of the initial velocity by the radius of the sphere of influence (see Equation (11)). The pseudocode below shows the function written for that purpose.

```
InitCraftsOnBodies()
       //calculate crafts unit velocity vector
       dirV = VectorDir(craft.V);

       //calculate the crafts initial position
       for (k=0; k<3; k++)
              offset = body.SOI*dirV[k];
              craft.R[k] = body.R[k]+offset;
```

Table 4 describes the variables used in InitCraftsOnBodies().

Table 4: Variables used in SLIIT's InitCraftsOnBodies() function

Variable	**Description**
VectorDir	A function which calculates the unit vector of a 3-D inputted vector
k	Counter to track 3 spatial dimensions
offset	A dummy variable to hold the offset direction from the planet's center in the current dimension
body.SOI	The radius of the planet's sphere of influence
craft.R	Craft position
body.R	Planet position

2.5 Programming Newton's Method

When the craft is launched from Earth on a specific date there are three independent variables which can be controlled (velocity in each spatial dimension). When the spacecraft arrives at Mars there are three distance functions which must equal zero (the distance to Mars in each spatial dimension). In addition, the ideal trajectory will have the spacecraft arriving at Mars such that the craft and Mars have co-linear velocity vectors. Therefore, the system of equations to be solved is

$$\begin{cases} X_{final}\left(v_{x_{initial}}, v_{y_{initial}}, v_{z_{initial}}\right) = 0 \\ Y_{final}\left(v_{x_{initial}}, v_{y_{initial}}, v_{z_{initial}}\right) = 0 \\ Z_{final}\left(v_{x_{initial}}, v_{y_{initial}}, v_{z_{initial}}\right) = 0 \\ \Theta_{final}\left(v_{x_{initial}}, v_{y_{initial}}, v_{z_{initial}}\right) = 0 \end{cases} \tag{35}$$

where X_{final}, Y_{final}, and Z_{final} are the distance between the spacecraft and Mars in each spatial dimension. Θ_{final} is the approach angle, defined as the angle between the velocity vectors at the point where they are closest in their given trajectories. The approach angle is calculated by solving the geometric definition of the dot product for Θ.

$$\boldsymbol{v_{craft}} \cdot \boldsymbol{v_{planet}} = \left|\boldsymbol{v_{craft}}\right|\left|\boldsymbol{v_{planet}}\right| cos\,\Theta$$

$$\Theta = cos^{-1}\left(\frac{\boldsymbol{v_{craft}} \cdot \boldsymbol{v_{planet}}}{\left|\boldsymbol{v_{craft}}\right|\left|\boldsymbol{v_{planet}}\right|}\right) \tag{36}$$

The following pseudocode shows how these parameters are evaluated at each timestep. If the craft is closer to its destination planet than it has been previously then the parameters are saved.

```
Dist2Dest()
       veldot=0;
       for (k=0; k<3; k++)
              vect[k]=craft.R[k]-body.R[k];
```

```
        veldot += craft.V[k]*body.V[k];

dist=VectorMag(vect);
veldot = veldot / ( VectorMag(craft.V) * VectorMag(body.V) );
angle = acos(veldot);

if (dist < craft.dist2dest)
        craft.dist2dest=dist;
        craft.vect2dest[k]=vect[k];
        craft.velangle=angle;
```

Table 5: Variable used in SLIIT's Dist2Dest() function

Variable	**Description**
k	Counter to track 3 spatial dimensions
dist	A dummy variable to hold the distance between the craft and the destination planet at the current timestep
vect	A dummy vector to hold the vector between the craft and the destination planet at the current timestep
angle	A dummy variable to hold the angle between the craft and the destination planet's velocity vectors at the current timestep
veldot	A dummy variable to hold the dot product of the velocities of the craft and the destination planet at the current timestep
craft.R	Craft position
craft.V	Craft velocity
body.R	Destination planet position
body.V	Destination planet velocity
craft.vect2dest	The vector between the craft and the destination planet at their closest
craft.dist2dest	The magnitude of craft.vect2dest
craft.velangle	The approach angle between the craft and planet at their closest

The iterative root finding technique requires a stopping criterion to be established. For this project it was decided that the magnitude of the vector between the spacecraft and the destination planet must be less than the planet's sphere of influence and that Θ_{final} must be less than 0.1 radians. Mars' sphere of influence was calculated in the same way as Earth's (see Equation (11)).

To use Newton's method to iteratively solve for the roots of a function it is necessary to calculate the Jacobian of the craft's trajectory. To approximate the derivatives of the system in

equation (30), three fictitious crafts are created when SLIIT is executed, each with slightly different initial velocities in each dimension. The original craft and its three "derivative" crafts are run in parallel for the duration of the trial. A matrix is built from the final values of each crafts' "vect2dest" and "velangle" parameters. Again, these parameters represent the relative position of the craft to its destination planet at the point in its trajectory when they are closest.

```
ValuesMatrix()
	//first column is original value, following columns are derivatives
	// first 3 functions are distance from destination planet in each dimension
	for (i=0; i<3; i++)
		values[i][0] = craft_0.vect2dest[i];
		values[i][1] = craft_dx.vect2dest[i];
		values[i][2] = craft_dy.vect2dest[i];
		values[i][3] = craft_dz.vect2dest[i];

	// fourth function is the approach angle between the velocities (weighted)
	values[3][0]=craft_0.velangle*gamma;
	values[3][1]=craft_dx.velangle*gamma;
	values[3][2]=craft_dy.velangle*gamma;
	values[3][3]=craft_dz.velangle*gamma;
```

Table 6: Variables used in SLIIT's ValuesMatrix() function

Variable	**Description**
craft_0	Original craft
craft_dx	"Fictitious" craft with starting velocity offset by dx in the x-direction
craft_dy	"Fictitious" craft with starting velocity offset by dy in the y-direction
craft_dz	"Fictitious" craft with starting velocity offset by dz in the z-direction
.vect2dest[i]	The i^{th} component of the vector between the associated craft and the destination planet at their closest
.velangle	The approach angle between the associated craft and planet's velocity vectors at their closest
gamma	A weighting function (described below)

The ValuesMatrix() function above includes a weighting factor, γ, on Θ_{final}. This is necessary because no solution was found that solved the over-constrained system in Equation (35). In addition, the stopping criteria of distance to the planet in AU is about 5 orders of magnitude smaller than that of the approach angle in radians which causes each iteration to preferentially

search for a result which favors a small approach angle over a small distance. This was remedied by introducing γ which is defined as

$$\gamma = \begin{cases} 1 & \Theta_{final} \geq \epsilon \\ D & \Theta_{final} < \epsilon \end{cases} \quad (37)$$

where ϵ is the acceptance criteria for the approach angle and D is the logarithmic difference between the orders of magnitude. For example, if the distance to the planet is $a \times 10^{-5}$ AU and the approach angle is $b \times 10^{-2}$ radians then $\gamma = 10^{-3}$ and the Θ_{final} dependent term in the values matrix is $b \times 10^{-5}$. This forces the dependence on Θ_{final} to be on the same order as the distance terms if the approach angle is within the acceptable range but allows the approach angle to dominate otherwise. The pseudocode below shows the calculation of γ.

```
CalcGamma()
	gamma=1;
	if(craft.velangle < epsilon)
		gamma=craft.dist2dest/craft.velangle;
		gamma=floor( log10(gamma) );
		gamma=pow(10,gamma);
```

After building the matrix of values the Jacobian is calculated as shown below.

```
Jacobian()
	for (i=0; i<nrows; i++)
		for (j=0; j<(mcols-1); j++)
			jacobian[i][j]=(values[i][j+1]-values[i][0])/d[j];
```

Table 7: Variables used in SLIIT's Jacobian() function

Variable	**Description**
jacobian	The Jacobian matrix
values	Matrix of values built in the ValuesMatrix() function (described above)
nrows	Number of rows in "values" matrix
mcols	Number of columns in "values" matrix
d	A vector of differential elements, defined as [dx, dy, dz]

After the Jacobian is calculated, Equation (29) is followed to calculate the next-iteration initial velocity of the craft. This calculation needs several standard linear algebra operations to be programmed which are not discussed here for brevities sake. The pseudocode below shows how the next-iteration initial velocity is calculated.

```
CalcNextVel()
       jacob=Jacobian();                    //jacobian is nxm
       invjacob=MatPseudoInvert(jacob);     //invjacob is mxn
       nextguess=MatMultiply(invjacob,values1);

       for (i=0; i<m; i++)
              nextguess[i]=initvel[i]-nextguess[i];
```

Table 8: Variables used in SLIIT's CalcNextVel() function

Variable	**Description**
jacob	The Jacobian matrix
invjacob	The inverse of the Jacobian
values1	First column of the "values" matrix
n	Number of equations
m	Number of unknowns
MatPseudoInvert	A function which calculates the pseudoinverse of a matrix
MatMultiply	A function which multiplies two matrices
initvel	The initial-velocity vector for the current iteration
nextguess	The next-iteration initial-velocity vector

3 Results

The boundary conditions discussed in the earlier sections were written to a text-based input file and SLIIT was run. Figure 3.1 shows the resulting path of Earth, Mars, and the spacecraft using the initial conditions predicted as if the craft were to follow a Hohmann transfer. Although they are not shown in the plot, the gravitational influences of all of the planets in the solar system are considered. The initial guess led to a trajectory in which the spacecraft tailed significantly behind Mars. At the closest point in the trajectory, the spacecraft was 0.95 AU from Mars and had an approach angle of 0.91 radians. This point was 167 days after launch.

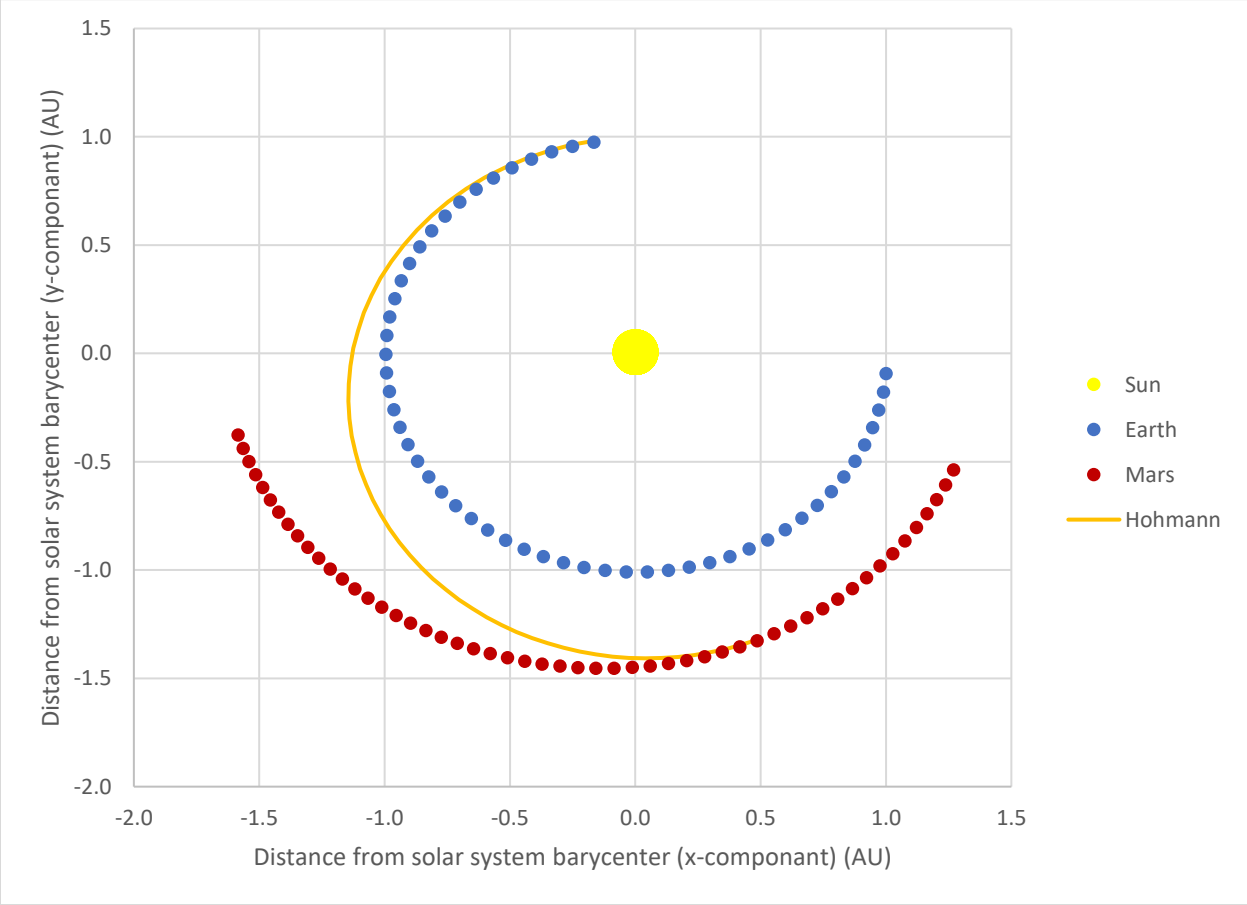

Figure 3.1: Trajectory from Earth to Mars using Hohmann transfer initial conditions.

More iterations were performed, with each trial using the results from the previous trial to guess at better initial conditions. Table 9 shows the initial velocity vector used for each trial.

Table 9: Initial velocities used for each iteration

Trial	x-component (AU/day)	y-component (AU/day)	z-component (AU/day)
Hohmann	-1.86×10^{-2}	-3.15×10^{-3}	0.00
Iteration 1	-1.74×10^{-2}	-6.04×10^{-3}	-7.90×10^{-4}
Iteration 2	-1.74×10^{-2}	-6.65×10^{-3}	-1.85×10^{-4}
Iteration 3	-1.75×10^{-2}	-6.37×10^{-3}	8.07×10^{-4}
Iteration 4	-1.74×10^{-2}	-6.26×10^{-3}	6.40×10^{-4}
Iteration 5	-1.74×10^{-2}	-6.31×10^{-3}	6.08×10^{-4}
Iteration 6	-1.74×10^{-2}	-6.35×10^{-3}	7.27×10^{-4}

Table 10 shows the results of each iteration. The sphere of influence of Mars is 3.8×10^{-3} AU, and we see that the correct trajectory to put the spacecraft within Mars' SOI was found after six iterations.

Table 10: Final distance from the craft to Mars for each iteration

Trial	Time of Flight (Days)	Closest Distance (AU)	Smallest Approach Angle (radians)
Hohmann	167	0.950	0.910
Iteration 1	263	0.092	0.016
Iteration 2	252	0.061	0.073
Iteration 3	221	0.041	0.200
Iteration 4	264	0.015	0.033
Iteration 5	264	0.009	0.035
Iteration 6	264	8.13×10^{-5}	0.062

Figure 3.2 shows the trajectory of the craft for each iteration.

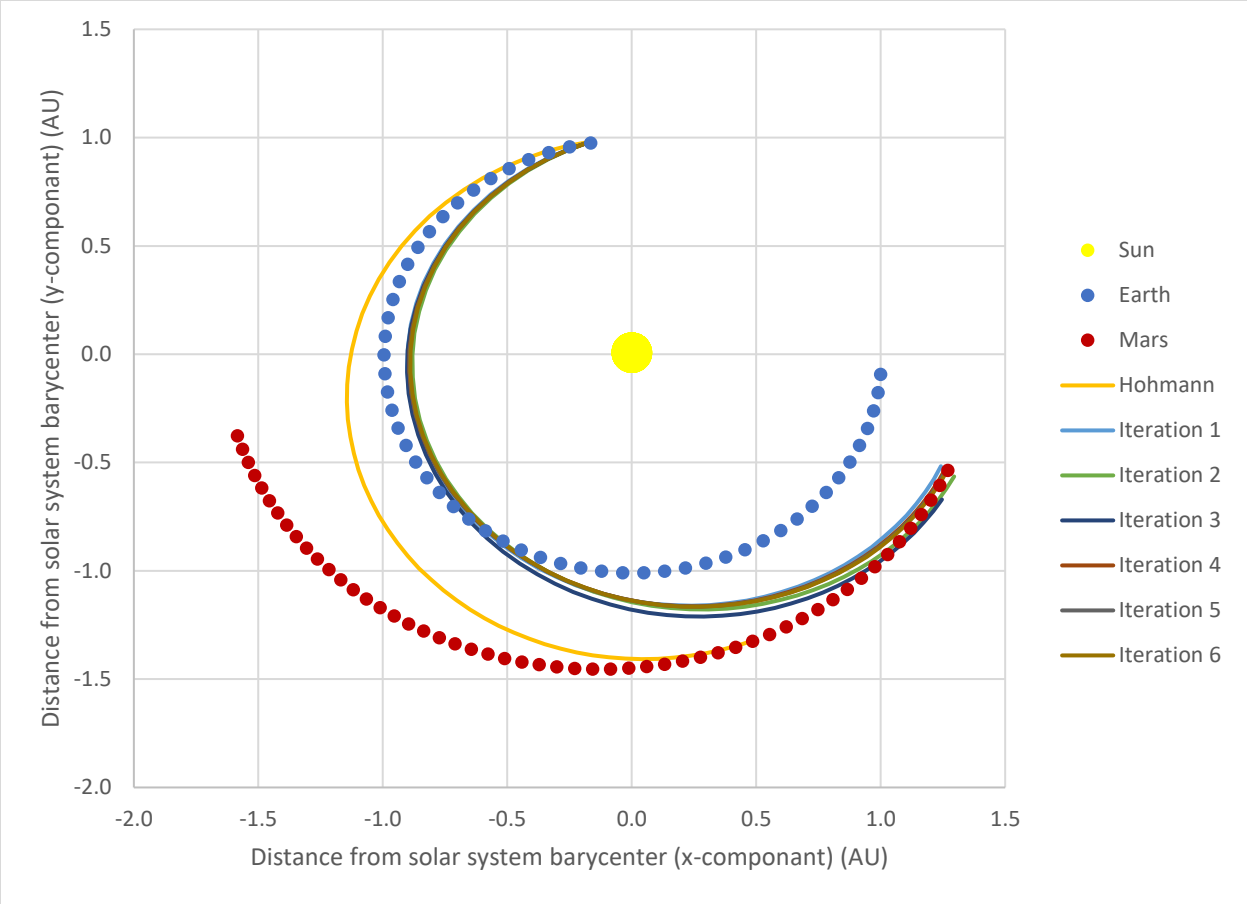

Figure 3.2: Successful planetary trajectory found after 6 iterations.

Figure 3.3 gives a closer look at the quadrant where the spacecraft arrives at Mars. We see significant improvement as early as the first iteration. The approach angle after the end of the first trial was larger than the acceptance criteria, so the Jacobian calculated was weighted to predict initial velocities which favored a smaller angle over shorter distances. The results after the first iteration had the distance reduced by an order of magnitude and the approach angle reduced by nearly two orders. Additional iterations all showed improvement in the distance, but the approach angle continued to vary since it is negatively weighted when less than the acceptance criteria.

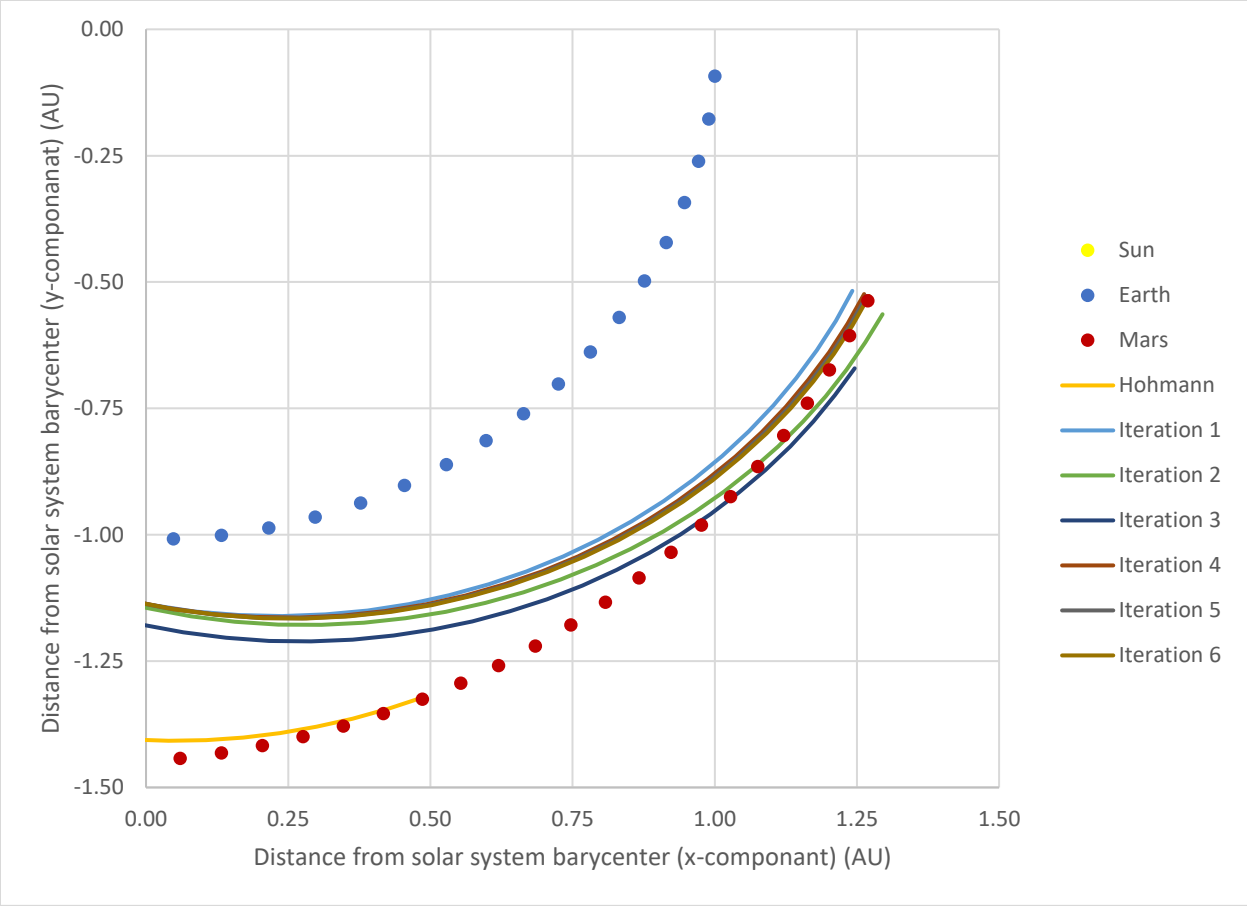

Figure 3.3: An enhanced view of the fourth quadrant of Figure 3.2 which shows the final portion of the trajectory for the six iterations leading to a successful transfer from Earth to Mars.

3.1 Timestep Sensitivity

As mentioned in the introduction, the advantage of using leapfrog integration over other is that it allows larger timesteps to be used. In the results shown above, the timestep was 0.01 days. To show the method's insensitivity to step size the trial was run again using the initial conditions from the successful transfer and a timestep of 0.001 days. The craft did not arrive at Mars on the first launch, but one of the derivative crafts did arrive allowing for a more ideal initial velocity to be calculated after only a single iteration. Table 11 shows the results from iteration 6 above and the results using a smaller timestep. Note that the smaller timestep led to final-

distance vector that is larger than the previous iteration. This is due to SLIIT being programmed to stop as soon as the craft is within Mar's SOI, and a larger timestep allows for the craft to get closer to Mars before the trial is terminated. These results support the results of others who have worked on this project [5].

Table 11: Final distance from the craft to Mars using a smaller timestep

Trial	Time of Flight (Days)	Closest Distance (AU)	Smallest Approach Angle (radians)
Iteration 6	264	8.13×10^{-5}	0.062
Small Step	264	8.58×10^{-5}	0.043

3.2 Comparison to ExoMars Spacecraft

The ExoMars spacecraft was launched from Earth on March 16, 2016. The methods described above are used to predict the flight path of that spacecraft in its trajectory to Mars. On that date the initial velocity for a Hohmann transfer is

$$\boldsymbol{v}_0 = \begin{bmatrix} -1.88 * 10^{-3} \\ -1.88 * 10^{-2} \\ 0.00 \end{bmatrix} \frac{AU}{day}. \qquad (38)$$

A trajectory which successfully arrived at Mars was again found after six iterations, as can be seen in Table 12.

Table 12: Final distance from the craft to Mars for each iteration from the ExoMars launch-date

Trial	Closest Distance (AU)	Smallest Approach Angle (radians)
Hohmann	0.300	0.031
Iteration 1	0.295	0.004
Iteration 2	0.169	0.157
Iteration 3	0.009	0.089
Iteration 4	0.0005	0.099
Iteration 5	0.007	0.090
Iteration 6	7.96×10^{-5}	0.131

Figure 3.4 shows the initial Hohman transfer trajectory, the sixth iteration which arrived at Mars, and the trajectory of the EXOMARS GTO spacecraft. We see that the trajectory calculated follows the MARS GTO spacecraft very closely initially then diverges slightly before reaching Mars. This divergence is due to a maneuver the GTO spacecraft performed during the course of its transfer which is not duplicated in the simulated trajectory.

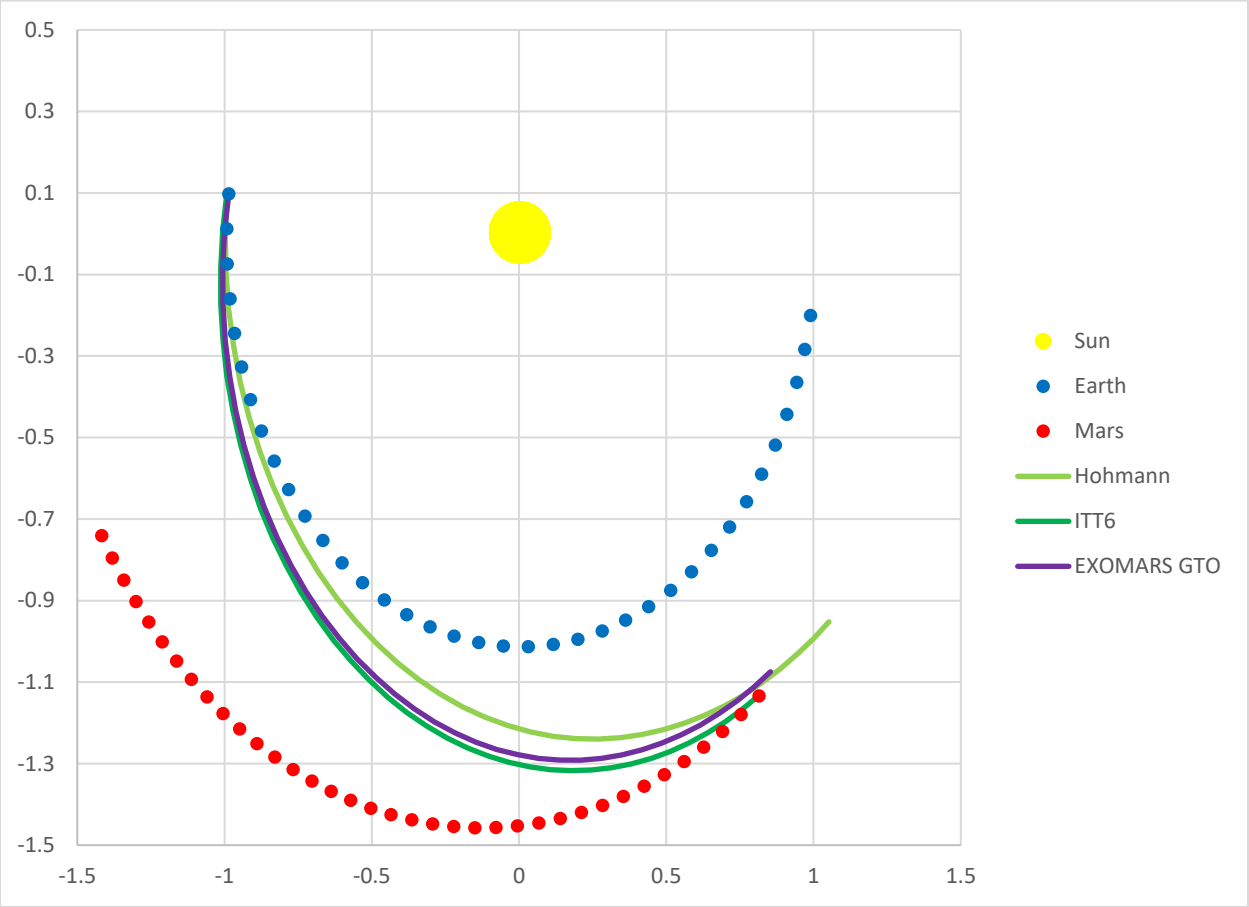

Figure 3.4: EXOMARS GTO trajectory compared to the trajectory predicted by SLIIT, the program written for this project. The sixth iteration (which successfully arrived at Mars) followed the path of the EXOMARS GTO craft very well for the first part of the trajectory. However, the EXOMARS GTO craft had a burn partway through the trajectory which lead to the divergence of the paths seen.

4 Analysis and Discussion

Figure 4.1 plots the results displayed in Table 10 on a semi-logarithmic scale. The initial trial, which chose initial conditions assuming a Hohmann transfer, missed the target planet by nearly 1 AU. This shows that while the Hohmann transfer got the craft going in the correct direction, it does not give realistic results. Each iteration improved the distance vector and the approach angle was maintained below the programmed minimum after the third iteration. The "closest distance" curve converges exponentially (linearly on the semi-logarithmic scale with an R^2 value of 0.84).

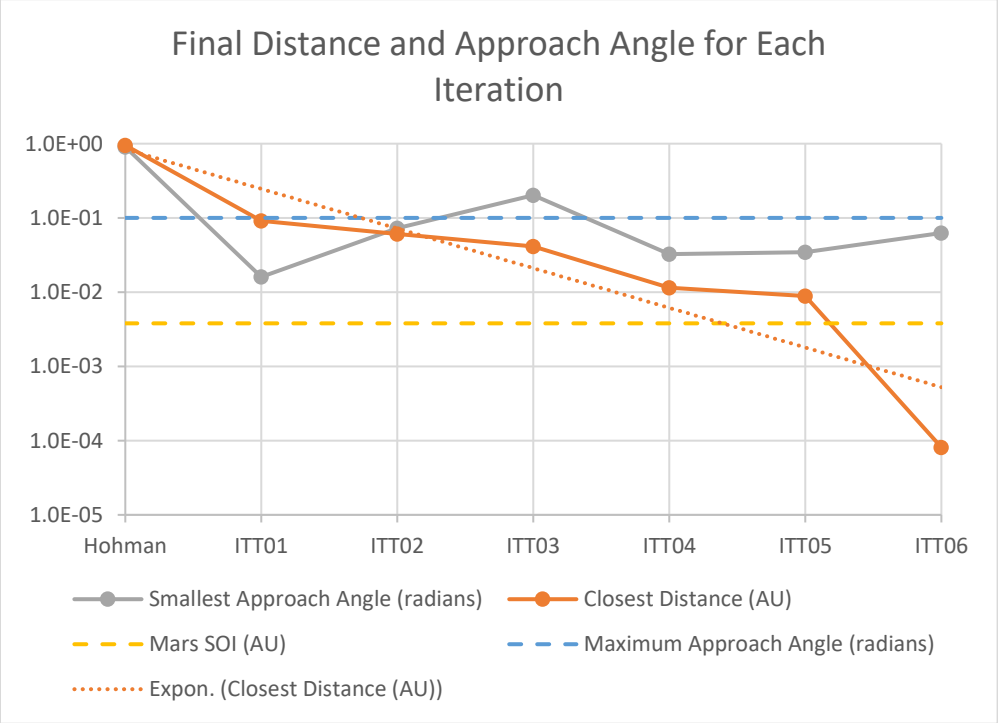

Figure 4.1: Final distance and approach angle for each iteration of a successful transfer from Earth to Mars on a semi-logarithmic scale. The orange and grey solid lines represent the distance and approach angle of the spacecraft at its closest distance to Mars. The dashed lines show the acceptance criteria; Mars' sphere of influence for distance and 0.1 radians for approach angle. The orange dotted line shows an exponential (linear on the semi-log plot) line of best fit for the closest distance.

The timestep sensitivity discussed in Section 3.1 shows that if the timestep is decreased by an order of magnitude, only a single additional iteration is required. This shows that the timestep used while iterating toward a solution was sufficiently small that the method used is insensitive to the size of timestep chosen; a ten-times-smaller timestep led to a relative error in the final position of the craft of 1.78×10^{-5}.

4.1 Further Work

A limitation of the SLIIT program is that it uses a launch date determined by assuming a Hohmann transfer. In the future the launch date could be a variable which is iterated on just as initial velocities are now. One advantage of this could be that it would make the Jacobian a square matrix and the system of equations would not be over-constrained, which could allow for a more precise solution.

Additionally, as SLIIT is currently written each iteration requires the input deck to be edited and for the user to launch the program manually. This method was chosen because one of the purposes of the project is to develop a learning tool it is valuable for the user to be forced to be involved in the process used to find solutions. However, solutions could be found much faster if the iterative process was automated, especially if the user is seeking a more complicated trajectory which may require many more than six iterations.

5 Conclusion

It has been shown that the leapfrog integration method combined with Newton's method of iterative root finding and reasonable initial conditions can be used to find the ideal trajectory of a spacecraft on an interplanetary trajectory. This trajectory considers the gravitational influence of all planets in the solar system using tabulated data of the planets' positions and seeks a solution which favors the least amount of energy. Newton's method of iterative root finding converges exponentially, and leapfrog integration allows for large timesteps which causes the program to find solutions quickly.

While this study only explored two Earth to Mars trajectories, the methods described can be used to calculate the ideal trajectory between any two planets in the solar system (including Earth's moon). Additionally, the program is written such that a user could input the absolute initial position and velocity of a craft, allowing them to model more complicated orbits or trajectories which do not originate on a planet.

6 Acknowledgments

This work is a continuation of Sam McLain's 2015 senior thesis under the advisement of Dr. Geibultowicz titled "Leapfrog Integration as an Accurate and Uncomplicated Alternative for N-Body Simulations in Computational Astronomy" [5]. This paper showed that the leapfrog integration scheme is a valid alternative for creating N-body simulations by demonstrating that this scheme is capable of providing position data of similar accuracy to higher-order methods.

7 Bibliography

[1] Aarseth, S. J. et al. (Eds.), The Cambridge N-Body Lectures (Lecture notes in physics, 760, p. 51), London: Springer-Verlag Berlin Heidelberg, 2008.

[2] Berrut, J. & Trefethen, L., "Barycentric Lagrange Interpolation," *Society for Industrial and Applied Mathematics,* vol. 46(3), pp. 501-517, 2004.

[3] Bettinger, R. A. & Black, Jonathan T., "Mathematical relation between the Hohmann transfer and continuous-low thrust Maneuvers," *Acta Astronautica,* vol. Vol.96, pp. pp.42-44, March-April 2014.

[4] NASA, "HORIZONS Web-Interface," Site Manager: R. S. Park, [Online]. Available: https://ssd.jpl.nasa.gov/horizons.cgi. [Accessed 18 April 2016].

[5] "Leapfrog Integration as an Accurate and Uncomplicated Alternative for N-Body Simulations in Computational Astronomy", S. McLain, Undergraduate senior thesis, Oregon State University Department of Physics, 2015.

8 Appendix A: SLIIT Input File

The SLIIT program reads the simulation parameters from a specifically formatted text file labeled '*trialname'*.inp. SLIIT is run from a command terminal with the only inline argument being the name of the input file with no extension. As it runs the input parameters and status of the trial are printed in the terminal and to an output file. When the trial concludes the next-iteration initial velocity is provided and a plot file is generated.

A sample input deck is displayed below.

```
*SLIIT input file
*The order of the input DOES matter
*Comments start with an asterix (*) as the first non-whitespace character.
*
*****************Simulation parameters******************
*InitialTime (JDCT)
 2458119. *noon on Dec 31, 2017
*TrialDuration (days)
 300
*Timestep (days)
 1.0E-2
*PrintFrequency (days)
 5 *days
*
***************Craft Properties**************************
**Initial position can be either the name of a body listed below or explicit cartesian
**coordinates wrt solar system barycenter.
**Enter 0 for unused values
*num_crafts
 3
*Name           Mass(kg)
 Hohman         1.0
*InitialPlanet  DestinationPlanet
 Earth          Mars
*InitPos.x      InitPos.y       InitPos.z    (initial position in AU)
 0.0            0.0             0.0
*InitVel.x      InitVel.y       InitVel.z    (initial velocity in AU/day)
 -1.86e-02     -3.15e-03      0.0
*InitAccel.x    InitAccel.y     InitAccel.z  (initial acceleration in AU/day/day)
*NOTE: Initial acceleration is not currently being used
 0.0            0.0             0.0
*
*Name           Mass(kg)
 ITT01          1.0
*InitialPlanet  DestinationPlanet
 Earth          Mars
*InitPos.x      InitPos.y       InitPos.z    (initial position in AU)
 0.0            0.0             0.0
*InitVel.x      InitVel.y       InitVel.z    (initial velocity in AU/day)
 -1.742173e-02 -6.043623e-03  -7.901637e-04
```

```
*InitAccel.x    InitAccel.y     InitAccel.z  (initial acceleration in AU/day/day)
*NOTE: Initial acceleration is not currently being used
 0.0            0.0             0.0
*
*Name           Mass(kg)
 ITT02          1.0
*InitialPlanet  DestinationPlanet
 Earth          Mars
*InitPos.x      InitPos.y       InitPos.z    (initial position in AU)
 0.0            0.0             0.0
*InitVel.x      InitVel.y       InitVel.z    (initial velocity in AU/day)
 -1.742204e-02 -6.645807e-03  -1.850269e-04
*InitAccel.x    InitAccel.y     InitAccel.z  (initial acceleration in AU/day/day)
*NOTE: Initial acceleration is not currently being used
 0.0            0.0             0.0
*
***************Bodies with known position files******************
*num_bodies
 9
*Name      plt    Mass(kg)        SOI(km)   PositionDataFile
Sun        1      1.988544E30     0.0       sun_bary_vectors.txt
Mercury    0      3.302E23        0.112E6   mercury_bary_vectors.txt
Venus      0      48.685E23       0.616E6   venus_bary_vectors.txt
Earth      1      5.97219E24      0.924E6   earth_bary_vectors.txt
Mars       1      6.4185E23       0.576E6   mars_bary_vectors.txt
Jupiter    0      1898.13E24      48.2E6    jupiter_bary_vectors.txt
Saturn     0      5.68319E26      54.6E6    saturn_bary_vectors.txt
Uranus     0      86.8103E24      51.8E6    uranus_bary_vectors.txt
Neptune    0      102.41E24       86.8E6    neptune_bary_vectors.txt
*
```