Simulations of Linear Proportional Navigation Engagements

Benjamin Parry

June 2023

Florida State University

Contents

1 Description of the Mathematics: 2

2 Description of the Program: 6

3 Trajectory Graphs: 9

4 Initial Conditions and Interception Data: 13

1 Description of the Mathematics:

For a linear engagement model, the path of the target is determined by its initial position, velocity and a constant acceleration. From these values, the engagement path of the pursuer is determined. This is done by taking advantage of proportional navigation and establishing a collision triangle. Using the following kinematic equations for the target and the pursuer, and forward Euler time stepping method, this is done.

Kinematics of the Target:

Due to the linear nature, the target's path is constructed using only three values (initial position, velocity, and flight path angle β) and their respective time derivatives (which are needed for the forward Euler method).

Given the initial velocity magnitude, the velocity components can be found using cos for x and sin for y, respectively:

$$V_{TX} = -V_T\, cos\, \beta \qquad V_{TZ} = V_T\, sin\, \beta$$

The established relationship between position, velocity and acceleration ($x(t)\frac{d}{dt} = v(t)$, $v(t)\frac{d}{dt} = a(t)$) are used to find $\dot{V}$ and $\dot{R}$:

$$\dot{V}_{TX} = a_T\, sin\, \beta \qquad \dot{V}_{TZ} = a_T\, cos\, \beta$$

$$\dot{R}_{TX} = V_{TX} \qquad \dot{R}_{TZ} = V_{TZ}$$

The time derivative of the target flight path angle β is proportional to the the target's acceleration and velocity magnitudes:

$$\dot{\beta} = a_T/V_T$$

Kinematics of the Pursuer:

Starting with an initial position, velocity and heading error (HE), the necessary kinematics are found.

Once again, the time derivative or position is given by velocity, and the time derivative of velocity is given by acceleration:

$$\dot{R}_{MX} = V_{MX} \qquad \dot{R}_{MZ} = V_{MZ}$$

$$\dot{V}_{MX} = -a_{MZ}^{True}\, sin\, \lambda \qquad \dot{V}_{MZ} = a_{MZ}^{True}\, cos\, \lambda$$

In this case, a_M^{True} is the true proportional navigation needed to create a collision triangle, defined as:

$$a_M^{True} = N\, V_c\, \dot{\lambda}$$

Relative Position and Velocity:

A vector measuring the distance between the pursuer and target, called relative position, and a vector measuring the difference in velocity, called relative velocity, is established by:

$$\vec{R}_{TM} = \vec{R}_M - \vec{R}_T$$

$$\vec{V}_{TM} = \vec{V}_M - \vec{V}_T$$

and with and components:

$$R_{TMX} = R_{MX} - R_{TX} \qquad R_{TMZ} = R_{MZ} - R_{TZ}$$

$$V_{TMX} = V_{MX} - V_{TX} \qquad V_{TMZ} = V_{MZ} - V_{TZ}$$

Closing Velocity:

Using relative position and velocity, a closing velocity, or the velocity needed to intercept the target, can be calculated as follows:

$$V_c = -\frac{d}{dt}\|\vec{R}_{TM}\| = -\frac{d}{dt}\left[\left((R_{TMX})^2 + (R_{TMZ})^2\right)^{\frac{1}{2}}\right]$$

using the chain rule, V_c becomes:

$$V_c = -\frac{R_{TMX}\,V_{TMX} - R_{TMZ}\,V_{TMZ}}{\|\vec{R}_{TM}\|}$$

Line of Sight Angle and Rate:

In addition, a line of sight angle, λ, defined as the angle between the horizon and the relative position vector, as well as its time derivative can be calculated:

$$\tan(\lambda) = \frac{R_{TMZ}}{R_{TMX}} \qquad \Rightarrow \qquad \lambda = \arctan\left(\frac{R_{TMZ}}{R_{TMX}}\right)$$

with a time derivative:

$$\frac{d}{dt}\lambda = \frac{d}{dt}\left[\arctan\left(\frac{R_{TMZ}}{R_{TMX}}\right)\right]$$

which by the chain rule becomes:

$$\dot{\lambda} = \frac{R_{TMX}\,V_{TMZ} - R_{TMZ}\,V_{TMX}}{\|\vec{R}_{TM}\|^2}$$

Lead Angle and Heading Error:

With the line of sight angle defined, the angle needed to complete the collision triangle, or lead angle L, can be formulated using law of sines:

$$\frac{V_T}{\sin(L + HE)} = \frac{V_M}{\sin(\beta + \lambda)}$$

$$\rightarrow \quad \sin(L + HE) = \frac{\sin(\beta + \lambda)}{V_M}\,V_T$$

$$\therefore \quad L = \arcsin\left(V_T\,\frac{\sin(\beta + \lambda)}{V_M}\right) - HE$$

Lastly, with all the necessary pursuer angles defined, the velocity components can be found by the following equations:

$$V_{MX} = V_M\,cos\,(\lambda + HE + L) \qquad V_{MZ} = V_M\,sin\,(\lambda + HE + L)$$

Forward Euler method:

Given a first order derivative in time:

$$\frac{dy}{dt} = f(t, y)$$

The derivative can be replaced with an approximation based on a forward difference derivative:

$$\frac{y(t+h)-y(t)}{h} \approx f(t,y)$$

Now discretize the equation. That means replace y(t) with a sampled function y_n defined on discrete times t_n. That is, $y_n = y(t_n)$. We also let h be small time steps, $h = t_{n+1} - t_n$, and the equation becomes:

$$\frac{y_{n+1}-y_n}{h} \approx f(t_n, y_n)$$

Replacing the $\approx$ sign with $=$ for convenience and reordering, the simplified equation forward in time is given by:

$$y_{n+1} = y_n + h\, f(t_n, y_n)$$

That is, the future value in time y_{n+1} can be found using known current value y_n, time step h and the time derivative $f(t_n, y_n)$. This method will be implemented for each state of both the target and pursuer.

Proportional Navigation and Collision Triangle:

Proportional Navigation (PN or ProNav):

If γ is the flight path angle, and λ is the line of sight angle (LOS), then a relationship between their time rates and a positive gain constant N is defined as:

$$\dot{\gamma}(t) = N\,\dot{\lambda}(t)$$

$$\frac{d\gamma}{dt} = N\,\frac{d\lambda}{dt}$$

Employing the same forward difference derivative approximation used in the forward Euler method, the above equation can be rewritten as:

$$\frac{\gamma_{n+1}-\gamma_n}{h} = N\,\frac{\lambda_{n+1}-\lambda_n}{h}$$

$$\rightarrow \quad \gamma_{n+1} = N\,(\lambda_{n+1} - \lambda n) + \gamma_n \quad n = 0, 1, 2, ...$$

The idea here is that the pursuer can create a collision by maneuvering its flight path in response to the LOS rate by rotating the velocity vector to lead the target.

The pursuer will adjust its flight path in accordance with the equation given above until an angle is achieved that results in a collision triangle, or a situation in which the pursuer and target will intercept.

To create this outcome, the change in velocity needed is defined as:

$$\dot{V}_{MX} = -a_M^{True}\,\sin\lambda \qquad \dot{V}_{MZ} = a_M^{True}\,\cos\lambda$$

$$a_M^{True} = N\,V_c\,\dot{\lambda}$$

where in general, $2 < N < 5$.

Collision Triangle:

A collision triangle is an engagement situation in which if the target and pursuer have a constant velocity, then there exits some lead angle L that will cause the two objects to eventually intercept each other.

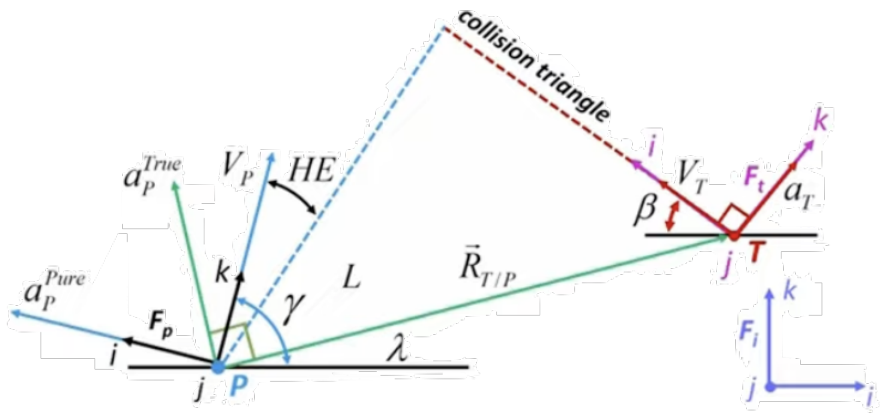
Because the velocities are constant, the paths form a triangle shape, and defined trigonometric properties can be used to find L:

$$\frac{V_M}{sin(\beta+\lambda)} = \frac{V_T}{sin(L+HE)}$$

$$\rightarrow \quad \sin\left(L+HE\right) = \frac{\sin\left(\beta+\lambda\right)}{V_M} V_T$$

$$\therefore \quad L = \arcsin\left(V_T \frac{\sin\left(\beta+\lambda\right)}{V_M}\right) - HE$$

Kinematics Diagram:



M	Missile		
T	Target		
V_M	Pursuer velocity mag	V_T	Target velocity magnitude
HE	Heading error	a_T	Target acceleration (const)
L	Lead angle	β	Target flight path angle
γ	Flight path angle	R_{TM}	Range to target (vector)
λ	Line of sight angle		
a_M^{True}	True PN acceleration cmd		

2 Description of the Program:

Define and initialize initial engagement conditions:

```
long double aT, double HE, int N, double h, long double beta_rad, long double RTX long double RTZ,

long double RMX, long double RMZ, long double VM, long double VT
```

Compute relative position components and magnitude:

```
long double R_TMX= RTX-RMX
long double R_TMZ= RTZ-RMZ
long double RTM= sqrt((R_TMX)ˆ 2 + (R_TMZ)ˆ 2)
```

Compute line of sight and lead angle:

```
long double lambda_r= arctan(R_RTMZ / R_TMX)
long double L= arcsin(VT * sin(beta_rad + lambda_r) / VM) - HE
```

Compute pursuer and target velocity components:

```
long double VMX= VM * cos(lambda_r + L + HE)
long double VMZ= VM * sin(lambda_r + L + HE)

long double VTX= -VT * cos(beta_rad)
long double VTZ= VT * sin(beta_rad)
```

Compute relative velocity components:

```
long double V_TMX= VTX - VMX
long double V_TMZ= VTZ - VMZ
```

Compute time derivative of LOS angle:

```
long double lambda_dot= ((R_TMX * V_TMZ) - (R_TMZ * V_TMX)) / (RTM)ˆ 2
```

Compute closing velocity:

```
long double VC= -((R_TMX * V_TMX) - (R_TMZ * V_TMZ)) / RTM
```

With all the necessary values defined and computed for time zero, forward time iteration is implemented using a while loop to find beta_rad, RTX, RTZ, RMX, RMZ, VTX, VTZ, VMX, and VMZ at the next time step (t+h), and recomputing relative position, relative velocity, LOS angle and rate, and closing velocity using the new (t+h) values.

This process is repeated until the relative position magnitude is within 0.1 meters, or in other words, until the pursuer and target come within 0.1 meters of each other. It is worth nothing that in order to reduce the number of computations and run time needed to satisfy this condition, the time step h decreases as the two objects get within certain distances of each other. This condition will be explained further when it is defined in the while loop.

Pseudo code for while loop:

Define while loop conditions and time variable:

```
bool intercept= false, bool ground= false, double t= 0
```

Define variables to store collision information in the case that the user wants to continue the loop after a collision is satisfied or for values $\mathbb{R}^{z<0}$:

```
double exit_time= 0
long double Txint, Tzint, Mxint, Mzint
```

while loop:

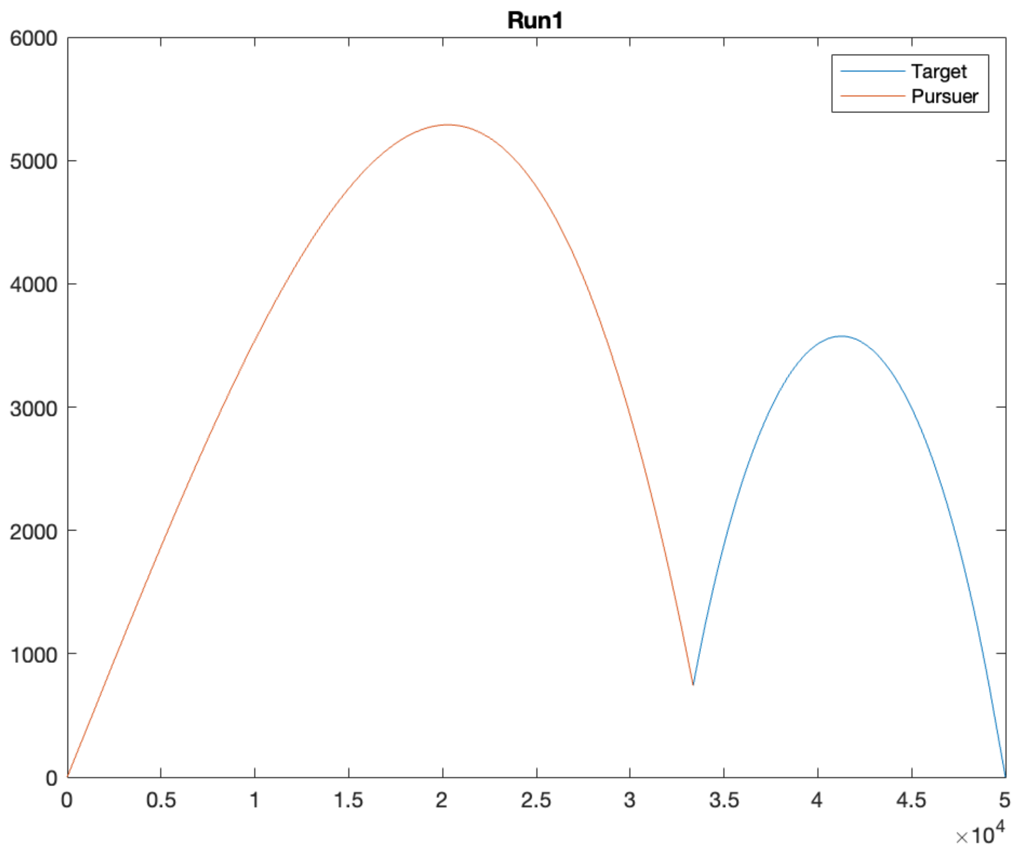
```
while(!intercept && !ground){
    t= t + h
    forward Euler method for state values (ie. $R_{i+1} = R_i + h\,\dot{R}$):
    beta_rad+= h * (aT / VT)
    RTX+= h * (-VT * cos(beta_rad))
    RTZ+= h * (VT * sin(beta_rad))
    RMX+= h * (VMX)
    RMZ+= h * (VMZ)
    VTX+= h * (aT * sin(beta_rad))
    VTZ+= h * (aT * cos(beta_rad))

    closing velocity and ProNav for pursuer velocity adjustements:
    nc= N * VC * lambda_dot
    VMX+= h * (-nc * sin(lambda_r))
    VMZ+= h * (nc * cos(lambda_r))

    recalculate values using t+h values:
    VT= sqrt((VTX)^ 2 + (VTZ)^ 2)

    R_TMX= RTX - RMX
    R_TMZ= RTZ - RMZ
    RTM= sqrt((R_TMX)^ 2 + (R_TMZ)^ 2)

    V_TMX= VTX - VMX
    V_TMZ= VTZ - VMZ

    lambda_r= arctan(R_TMZ / R_TMX)
    lambda_dot= ((R_TMX * V_TMZ) - (R_TMZ * V_TMX)) / (RTM)^ 2

    VC= -((R_TMX * V_TMX) - (R_TMZ * V_TMZ)) / RTM
```

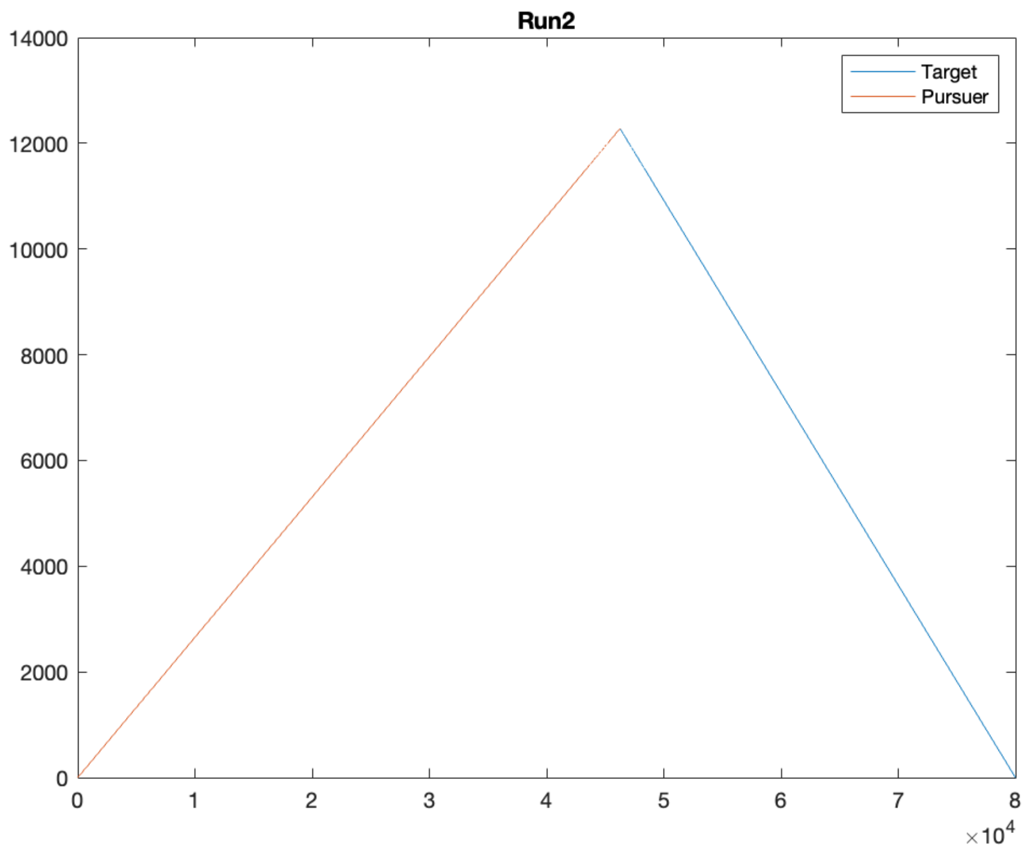
Even once a collision triangle has been established, the forward Euler method needs a small enough h that calculations will occur close enough to the true time of collision for the collision condition to be set to true. In other words, because a non continuous equation replaces the original continuous equation in the Euler method, if the time step is not small enough to have a calculation occur at the actual intersection (or close enough to satisfy RTM <= 0.1) then the program will not recognize that the two objects collided. One solution is to just set h to a small enough value that this is certain to occur, but the problem with that is if the time to intersection is large then it requires an enormous number of calculations to reach. So as a solution, different h values are assigned depending on how close the pursuer and target are to each other (RTM value). An initial value of 1 second is assigned, then incremental distances ranges are assigned to decrease h as the two objects grow nearer, until they are close enough that h must be small enough:

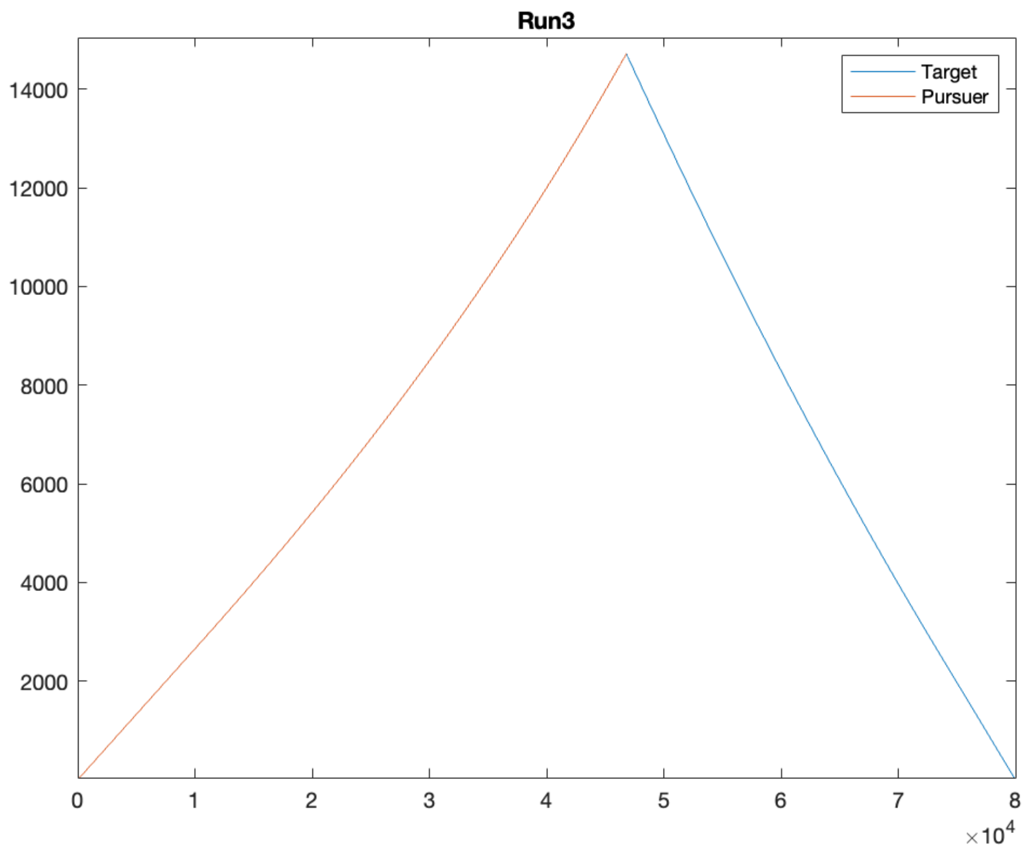
```
    if( RTM >= 100000)
        h=0.5
    else if( RTM < 100000 && RTM >= 50000)
        h=0.1
    else if( RTM < 50000 && RTM >= 10000)
        h=0.01
    else if( RTM < 10000 && RTM >= 5000)
        h=0.001
    else if( RTM < 5000 && RTM >= 1000)
        h=0.0001
    else if( RTM < 1000 && RTM >= 100)
        h=0.00001
    else if( RTM <= 0.1 ){
        exit_time= t
        Txint= RTX
        Tzint= RTZ
        Mxint= RMX
        Mzint= RMZ
        intercept= true
    }

    if( RTZ < 0 —— RMZ < 0){
        exit_time= t
        ground= true
    }
}
```

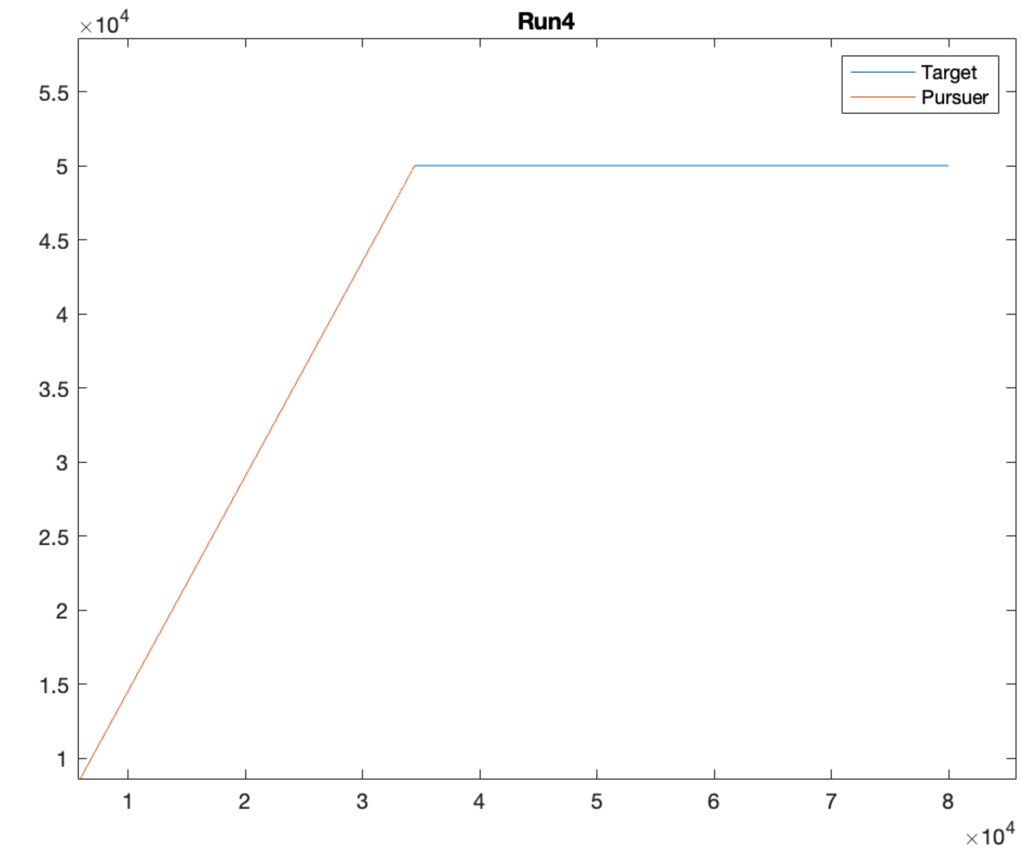
It is worth noting that the ranges chosen for the incremental h values, are large and with some trial and error more efficient ranges can be found to further reduce run time and total computations needed.

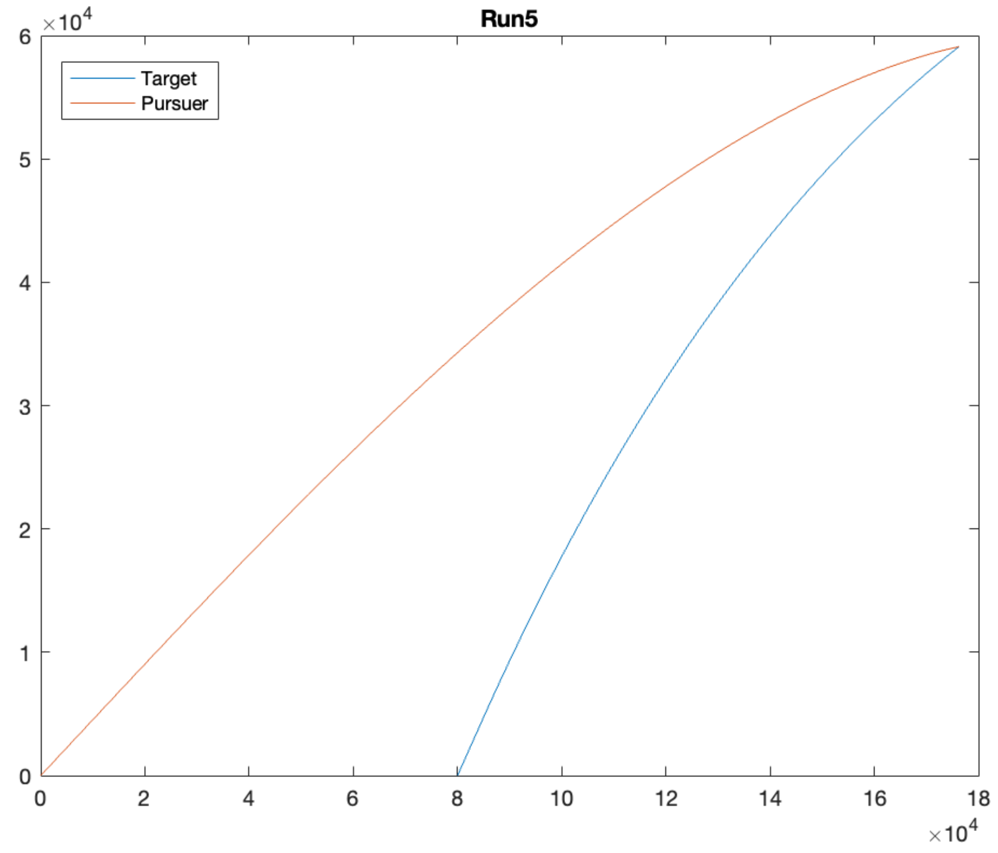
3 Trajectory Graphs:

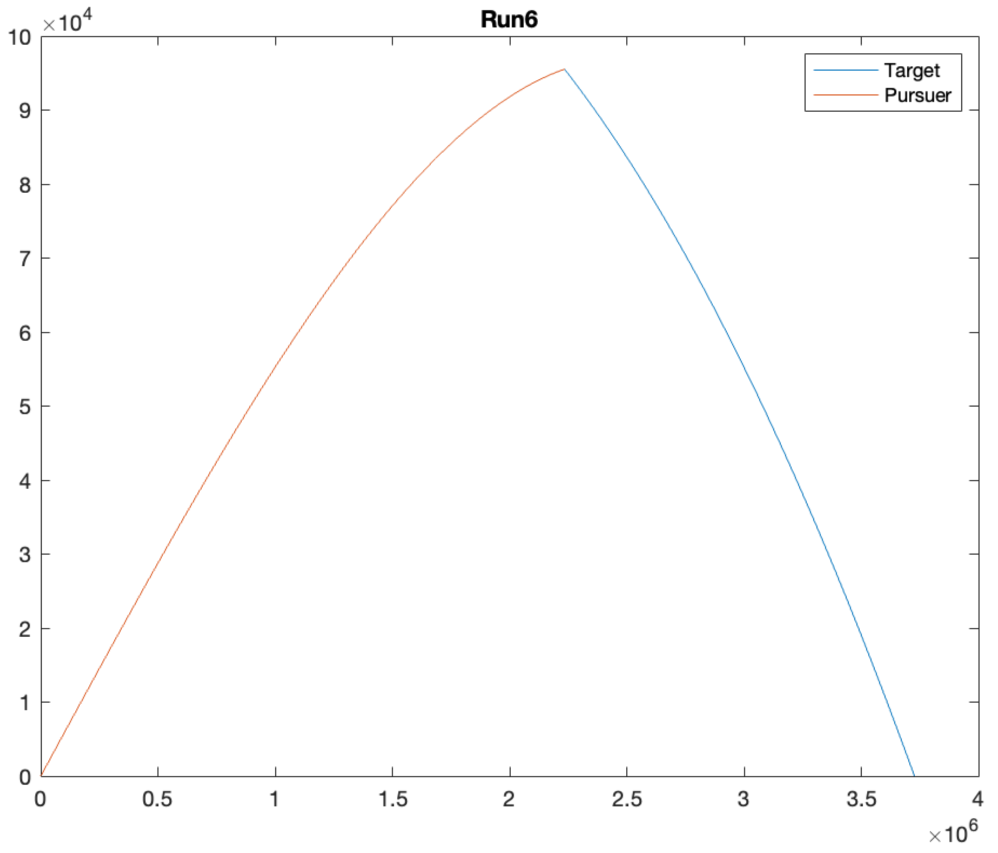


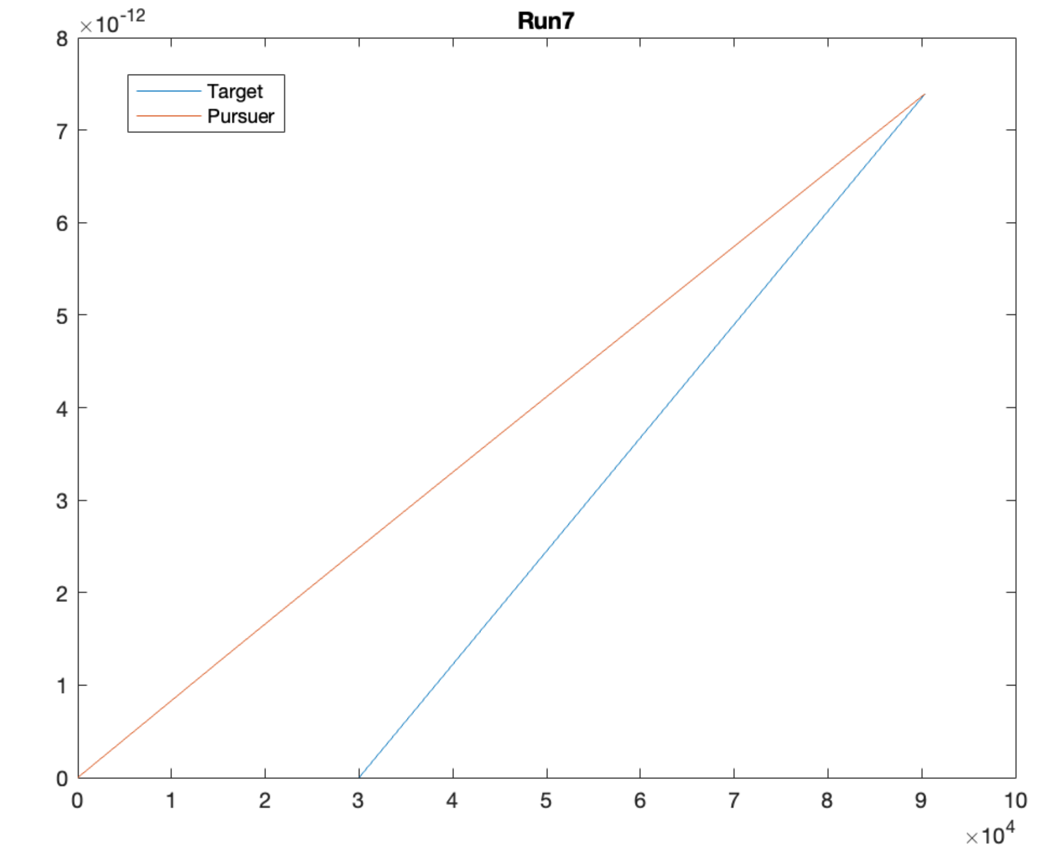


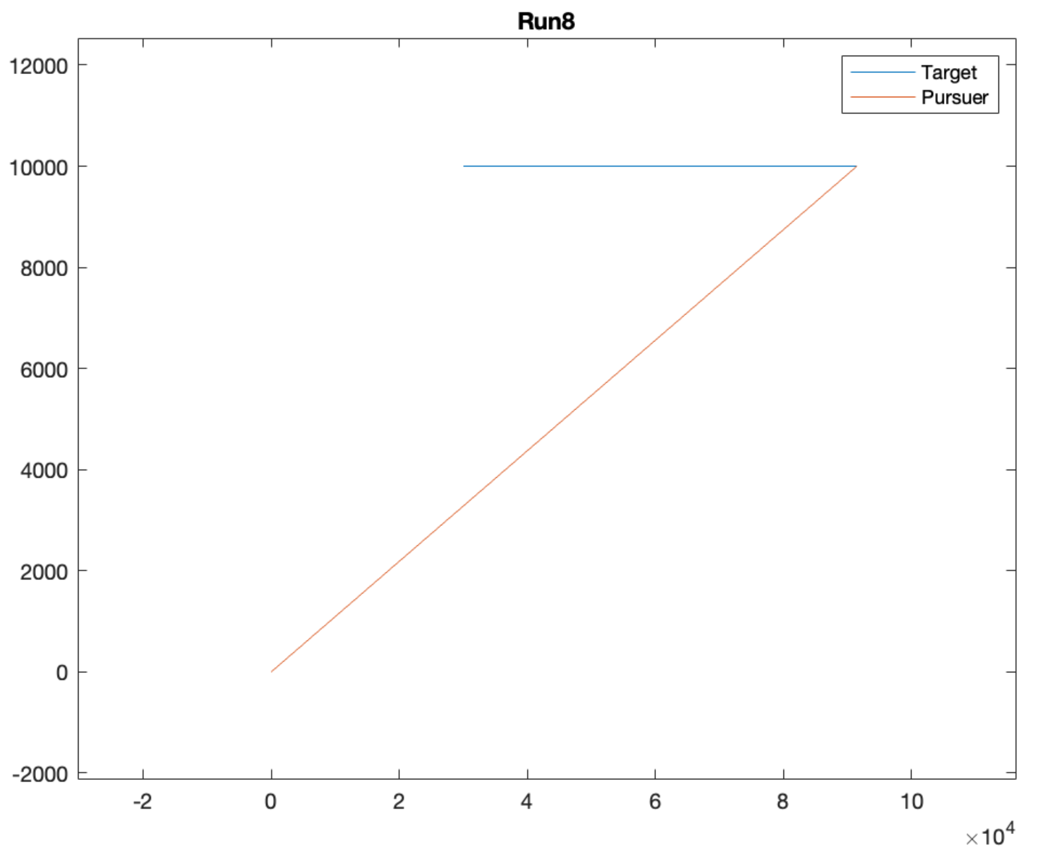

Run3
Target
Pursuer
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
$\times 10^4$


Run4
$\times 10^4$
Target
Pursuer
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
1
2
3
4
5
6
7
8
$\times 10^4$


Run5
Target
Pursuer
×10^4
0
1
2
3
4
5
6
0
2
4
6
8
10
12
14
16
18
×10^4


Run6
Target
Pursuer
×10^4
0
1
2
3
4
5
6
7
8
9
10
0
0.5
1
1.5
2
2.5
3
3.5
4
×10^6


Run7
×10^-12
Target
Pursuer
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
10
×10^4


Run8
Target
Pursuer
12000
10000
8000
6000
4000
2000
0
-2000
-2
0
2
4
6
8
10
×10^4

4 Initial Conditions and Interception Data:

Initial Condition Format:
$[a_T, HE, N, \beta, R_{TX}, R_{TZ}, R_{MX}, R_{MZ}, V_T, V_M]$

Run1:

Time taken by function: 471 milliseconds

Initial Conditions:
$[-80, 0.349066, 3, 0.785398, 50000, 0, 0, 0, 1000, 2000]$

Target was intercepted at time: 18.336760 seconds

Target position: (33358.877264, 741.257563)

Missile position: (33358.785036, 741.282097)

Run2:

Time taken by function: 57 milliseconds

Initial conditions:
$[0, 0.349066, 3, 0.349066, 80000, 0, 0, 0, 7500, 10000]$

Target was intercepted at time: 4.786660 seconds

Target position: (46265.081898, 12278.506044)

Missile position: (46264.994163, 12278.506044)

Run3:

Time taken by function: 51 milliseconds

Initial conditions:
$[200, 0.349066, 3, 0.349066, 80000, 0, 0, 0, 7500, 10000]$

Target was intercepted at time: 4.845730 seconds

Target position: (46806.341483, 14717.865247)

Missile position: (46806.323355, 14717.864573)

Run4:

Time taken by function: 53 milliseconds

Initial conditions:
$[0, 0.349066, 3, 0, 80000, 50000, 0, 0, 7500, 10000]$

Target was intercepted at time: 6.072330 seconds

Target position: (34457.525000, 50000.000000)

Missile position: (34457.510799, 49999.991124)

Run5:

Time taken by function: 325 milliseconds

Initial conditions:
$[200, 0.349066, 3, 2.356194, 80000, 0, 0, 0, 7000, 12000]$

Target was intercepted at time: 16.293070 seconds

Target position: (176278.966839, 59101.046540)

Missile position: (176278.882798, 59101.075334)

Run6:

Time taken by function: 92 milliseconds

Initial conditions:
$[-3.1245, 0.785398, 3, 0.087266, 3727000, 0, 0, 0, 10020, 15000]$

Target was intercepted at time: 149.244160 seconds

Target position: (2234764.830983, 95514.446966)

Missile position: (2234764.738533, 95514.447894)

Run7:

Time taken by function: 215 milliseconds

Initial conditions:
$[0, 0.174533, 3, 3.141593, 30000, 0, 0, 0, 10020, 15000]$

Target was intercepted at time: 6.024080 seconds

Target position: (90361.281600, 0.000000)

Missile position: (90361.200000, 0.000000)

Run8:

Time taken by function: 391 milliseconds

Initial conditions:
$[0, 0.174533, 3, 3.141593, 30000, 10000, 0, 0, 10020, 15000]$

Target was intercepted at time: 6.133530 seconds

Target position: (91457.970600, 10000.000000)

Missile position: (91457.877883, 9999.969094)

Program file (written in c++), copies of graphs, and the data files used to make the graphs can be found at:

https://github.com/benparry1