11th ANKARA INTERNATIONAL AEROSPACE CONFERENCE **AIAC-2021-176**
8-10 September 2021 - METU, Ankara TURKEY

SOLUTION OF THE MODIFIED POINT MASS TRAJECTORY MODEL USING OBJECT ORIENTED PROGRAMMING BY MODERN FORTRAN

Hüseyin Emrah Konokman [1]
TÜBİTAK SAGE
Ankara, Turkey

Mustafa Kaya [2]
Ankara Yıldırım Beyazıt University
Ankara, Turkey

ABSTRACT

Object oriented programming techniques using Modern Fortran are demonstrated for the solution of the modified point mass trajectory model. Fundamental concepts of this technique were demonstrated and the unified modeling language charts were given for the explicit integration methods including Euler and 4th order Runge-Kutta methods and for the modified point mass trajectory model. Using the developed code, trajectories of the 35 mm TP-T projectile were calculated.

INTRODUCTION

For the solution of the moving objects, one of the most general method is 6-degrees of freedom equations of motion model with point mass approximation. However, the solution of this model could be time consuming for the spin stabilized projectiles. Modified point mass trajectory model for the spin stabilized projectile, which is also given by STANAG 4355 Edition 3, 2009 was derived by [Lieske et.al (1966)]. This model is not as time-consuming to solve compared to 6-DOF model and it provides enough accuracy.

There are mainly two types of programming techniques: Procedural (procedure based) programming and object-oriented programming. Codes developed according to the procedure based programming are a bit faster than object-oriented ones but since object-oriented programming provides ease of further development and maintenance of the code and reuse of parts (classes) of the code, object-oriented programming technique becomes popular also in scientific computing.

For the solution of science and engineering problems, Fortran language was mostly used for years after 1950s. With Fortran 90 standard (after Fortran 90 it is also called as Modern Fortran) the modularity and derived types (encapsulation features) were added. Fortran 2003 included many object oriented programming (OOP) features such as type bound procedure and procedure pointer. Then 2008 standard provided the most of the important OOP features such as inheritance and polymorphism by adding extended type, abstract type and deferred binding, etc. Because Fortran language was (and is continuing to be) developed especially for the numerical calculations it is the fastest language for scientific calculations and numerical solutions compared to other languages. Learning, writing and reading it is easy (its syntax is simple), it is modern (it supports modularity, object-oriented programming and parallel

[1] Chief Research Engineer, Email: emrah.konokman@tubitak.gov.tr
[2] Assoc. Professor in Aerospace Engineering, Email: mukaya@ybu.edu.tr

programming guaranteed by its standard), it has a huge source of codes and libraries in open literature, and it is standard (it has an ANSI standard, which is updated according to the user needs and new modern techniques) and all compilers follow and apply the standardization rules.

There are four main principles of object oriented programming, which are encapsulation and information hiding, abstraction, inheritance and polymorphism. Encapsulation and information hiding is that all important information is contained inside the object, and only selected information is supplied to the outside. In a Fortran code, encapsulation is provided by “module” construct and derived types. “Private” statement is used to hide the information or procedures and “public” statement gives access to them from outside of the module. Abstraction is used to hide unnecessary details from the user providing the user to handle the complexity. Data abstraction is constituted by abstract data types and by derived types in a Fortran code [Metcalf (1995)]. Components of an object are all included in the derived type of that object and the structure for that object can be generated referring its derived type. Components of the object are encapsulated in its derived type. “Derived type” was first introduced by Fortran 90 and type bound procedure, which is one of the main steps through the object-oriented programming, was included by 2003 standard. Inheritance allows classes to inherit features of other classes [Educative (2021)], that is, it provides deriving new classes from an existing class using the common data and procedures. Inheritance enhances the reusability of the codes. In Fortran, “type extension” (extended type), which came with 2008 standard, provides the inheritance. By means of polymorphism concept, the object of different types can be accessed through the same interface [Stackify (2021)]. Each type can provide its own, independent implementation of this interface. Polymorphism concept is implemented using “abstract type” and “extend”ing this abstract type in Fortran, which was first introduced by 2008 standard.

The book of Rouson, D., Xia, J., Xu, X. (2011) explains object oriented design of the scientific software mainly using Fortran and also using C++. As this book states, two languages dominate the scientific computing, which are Fortran and C++. Simon Management Group (2006) gives an HPC survey stating that the United States Department of Defense High Performance Computing users had indicated that approximately 85% write in C/C++/C#, while nearly 60% write in Fortran. However, in recent years Fortran loses popularity among mechanical, civil and aerospace engineers in Turkey. Because of the ease of code development using present libraries, Matlab is very popular especially in the defence industry. Even Python has no much usage for computational codes. Although the code development may be much more easy and faster compared to the lower level languages Fortran and C++, since Matlab codes are much slower, the computational cost in terms of the time becomes excessive for the lifetime usage of these codes.

The aim of this study is the development of the numerical solution of the modified point mass trajectory model and demonstration of the object oriented programming techniques using Modern Fortran for this solution.

METHOD

Modified Point Mass Trajectory Model

Equation of motion of the projectile center of mass is given as equation (1).

$$m\dot{\mathbf{u}} = \mathbf{DF} + \mathbf{LF} + \mathbf{MF} + m\mathbf{g} + m\boldsymbol{\Lambda} \tag{1}$$

where $\mathbf{DF}$ is the drag force vector, $\mathbf{LF}$ is the lift force vector, $\mathbf{MF}$ is the Magnus force, $m\mathbf{g}$ is the weight and $m\boldsymbol{\Lambda}$ is the force vector due to Coriolis acceleration, and these are given in the following equations.

$$\mathbf{DF} = -\frac{\pi\rho d^2}{8}(C_{D0} + C_{D\alpha^2}{\alpha_e}^2)v \cdot \mathbf{v} \tag{2}$$

$$\mathbf{LF} = \frac{\pi\rho d^2}{8}(C_{L\alpha} + C_{L\alpha^3}{\alpha_e}^2)v^2 \cdot \boldsymbol{\alpha}_\mathbf{e} \tag{3}$$

$$\mathbf{MF} = -\frac{\pi\rho d^3 p C_{mag-f}}{8}(\boldsymbol{\alpha}_\mathbf{e} \times \mathbf{v}) \tag{4}$$

$$m\mathbf{g} = -mg_0 \begin{bmatrix} x_1/R \\ 1 - 2x_2/R \\ x_3/R \end{bmatrix} \quad (5)$$

$$m\mathbf{\Lambda} = 2\mathbf{m}(\boldsymbol{\omega} \times \mathbf{u}) \quad (6)$$

$$\mathbf{u} = \mathbf{u_0} + \int_0^t \dot{\mathbf{u}} dt \quad (7)$$

$$\mathbf{v} = \mathbf{u} - \mathbf{w} \quad (8)$$

where $\mathbf{x}$ is the position vector, $\mathbf{u}$ is velocity vector, $\mathbf{w}$ is the wind vector, v is the velocity vector with respect to the air, $\boldsymbol{\alpha_e}$ is vector of the yaw of repose. C_{D0}, $C_{D\alpha^2}$, $C_{L\alpha}$, $C_{L\alpha^3}$ and C_{mag-f} are the aerodynamic coefficients, which are the drag coefficient, induced drag coefficient, lift coefficient, cubic lift coefficient, Magnus force coefficient and spin damping coefficient, respectively.

Equation for the spin of the projectile is given as below.

$$\frac{d\mathbf{p}}{dt} = \frac{\pi \rho d^4 v C_{spin}}{8 I_x} \mathbf{p} \quad (9)$$

The yaw of repose vector is given as below.

$$\boldsymbol{\alpha_e} = -\frac{8 I_x v C_{spin}}{\pi \rho d^3 (C_{M\alpha} + C_{M\alpha^3} {\alpha_e}^2) v^4} \quad (10)$$

Additionally, position vector of the projectile with respect to the ground-fixed coordinate system is calculated as below.

$$\mathbf{x} = \mathbf{x_0} + \int_0^t \mathbf{u} dt \quad (11)$$

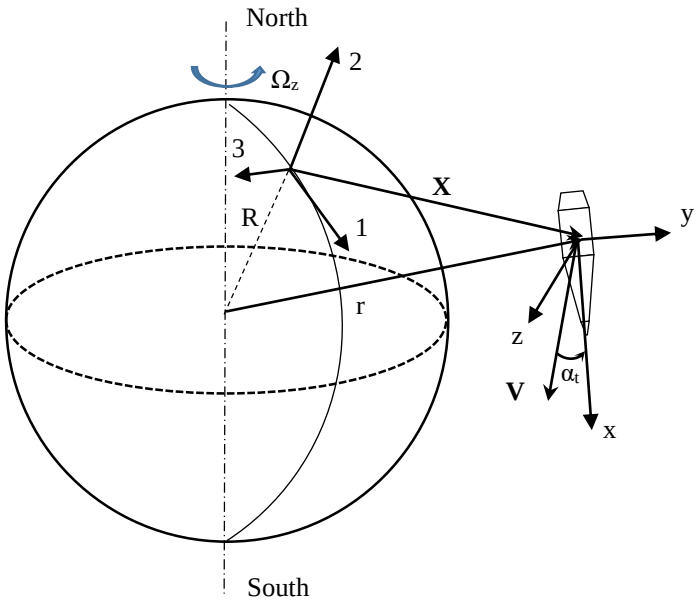


Figure 1. Ground Fixed Coordinate System for the Problem

For the ground fixed coordinate system as shown in Figure 1, the scalar form of equations of motion can be obtained as the following equations from equation (1).

$$\frac{du_1}{dt} = -\frac{\pi\rho d^2}{8m}\left[C_{D0} + C_{D_{\alpha^2}}(\alpha_e)^2\right]vv_1 + \frac{\pi\rho d^2}{8m}\left(C_{L\alpha} + C_{L_{\alpha^3}}{\alpha_e}^2\right)v^2\alpha_{e1} - g_0\frac{X_1}{R} - \frac{\pi\rho d^3 pC_{mag-f}}{8m}(\alpha_{e2}v_3 - \alpha_{e3}v_2) - 2\Omega[\sin(lat)u_3 + \cos(lat)\sin(Az)u_2] \quad (12)$$

$$\frac{du_2}{dt} = -\frac{\pi\rho d^2}{8m}\left[C_{D0} + C_{D_{\alpha^2}}(\alpha_e)^2\right]vv_2 + \frac{\pi\rho d^2}{8m}\left(C_{L\alpha} + C_{L_{\alpha^3}}{\alpha_e}^2\right)v^2\alpha_{e2} - g_0\left(1 - \frac{2X_2}{R}\right) - \frac{\pi\rho d^3 pC_{mag-f}}{8m}(\alpha_{e3}v_1 - \alpha_{e1}v_3) - 2\Omega[\cos(lat)\sin(Az)u_1 + \cos(lat)\cos(Az)\,u_3] \quad (13)$$

$$\frac{du_3}{dt} = -\frac{\pi\rho d^2}{8m}\left[C_{D0} + C_{D_{\alpha^2}}(\alpha_e)^2\right]vv_3 + \frac{\pi\rho d^2}{8m}\left(C_{L\alpha} + C_{L_{\alpha^3}}{\alpha_e}^2\right)v^2\alpha_{e3} - g_0\frac{X_3}{R} - \frac{\pi\rho d^3 pC_{mag-f}}{8m}(\alpha_{e1}v_2 - \alpha_{e2}v_2) - 2\Omega[\cos(lat)\cos(Az)\,u_2 - \sin(Az)u_1] \quad (14)$$

where

$$u_i = \frac{dX_i}{dt} \qquad i = 1,2,3 \quad (15)$$

Moreover, the scalar form of spin rate can be obtained from equation (9) as below.

$$\frac{dp}{dt} = \frac{\pi\rho d^4 pvC_{spin}}{8I_x} \quad (16)$$

The components of yaw of repose vector are obtained as below.

$$\alpha_{e1} = -\frac{8I_x p(v_2\dot{u}_3 - v_3\dot{u}_2)}{\pi\rho d^3(C_{M\alpha} + C_{M\alpha^3}{\alpha_e}^2)v^4} \quad (17)$$

$$\alpha_{e2} = -\frac{8I_x p(v_3\dot{u}_1 - v_1\dot{u}_3)}{\pi\rho d^3(C_{M\alpha} + C_{M\alpha^3}{\alpha_e}^2)v^4} \quad (18)$$

$$\alpha_{e3} = -\frac{8I_x p(v_1\dot{u}_2 - v_2\dot{u}_1)}{\pi\rho d^3(C_{M\alpha} + C_{M\alpha^3}{\alpha_e}^2)v^4} \quad (19)$$

Furthermore, v is the velocity of the projectile relative to air and its components are as below.

$$v_i = u_i - w_i \qquad i = 1,2,3 \quad (20)$$

where w is tthe wind velocity.

Additionally, the gravitational acceleration is obtained according to the position on Earth as $g_0 = 9.80665(1 - 0.0026\cos(2lat))\ m/s^2$, the angular speed of the Earth is $\Omega = 7.292115 \cdot 10^{-5}\ rad/s$, the radius of the Earth sphere approximating the geoid is $R_z = 6356766\ m$, lat is the latitude of the launch point of the projectile and Az is the azimuth of 1 axis.

Numerical Solution and Object Oriented Programming Techniques

For the numerical solution of a ordinary differential equations Euler method is the one of the simplest ones. Euler method is explicit type solution method, that is it uses variable values from known step. However, Euler method may give erroneous results for bigger steps.

4th order Runge-Kutta method is mostly used for the integration of the differential equations by many researchers since it gives relatively good results compared to the integration step size used. It is also explicit type method however it uses the equation values from the current time intermediate time step and further time step, so this method increases the accuracy. Since it is a common method, its formulation is not given in this paper again.

For the numerical integration of ordinary differential equations, a class called step_class was developed as a Fortran module and it is publicly shared from Github reposity by the author [Konokman (2021)]. The unified modeling language (UML) diagram of this class is shown by Figure 2 and the important parts of the code listing are given by Figure 3 (UML diagrams were drawn by means of ForUML software [Nanthaamornphong et al (2015)]). Differential equations set is given by a derived type and its deferred procedure. The code listing of the abstract type of the equation set is given by Figure 4. This class demonstrates the inheritance and polymorphism concepts. An abstract class (type) called step_abstract_type is first introduced. This abstract class includes the common variable ds (step size) and abstract interface for the step procedure, which takes vector (array) of equation set as input and gives vector of step of

the equation set as output. Figure 5 and Figure 6 shows the code listings for the Euler and 4th order Runge-Kutta methods which are the derived elements of the step class. These step methods are inherited from step_abstract_type class and “step” procedure for each new step class implemented via deferred binding. This is provided by Fortran using “deferred” keyword in “procedure” line of the type boung procedure of abstract type.

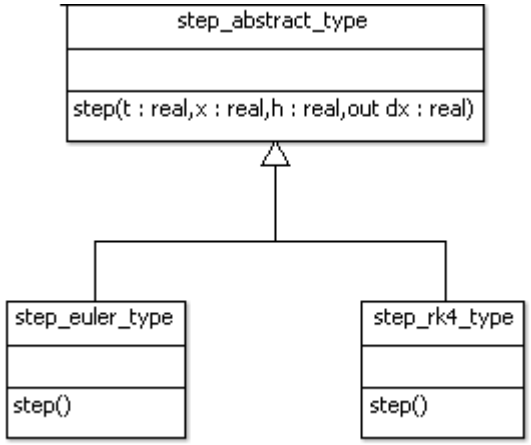


Figure 2. Step Class UML Diagram

```
!!!Abstract step type
!! The type for a new step method is extended from this abstract type
type, abstract, public :: step_abstract_type

contains
    procedure(step_proc), deferred, nopass :: step
end type step_abstract_type

abstract interface
    subroutine step_proc(f, t, x, h, dx)
        use :: precision, only: rp    !!!rp is the selected real precision, ie
                                      !! single, double or quadruple precision
        import :: derivative_abstract_type
        class(derivative_abstract_type) :: f !!!The derivative function container
        real(rp), intent(in) :: t            !!!Independent variable
        real(rp), intent(in) :: x(:)         !!!Vector of the equation set
        real(rp), intent(in) :: h            !!!Step size
        real(rp), intent(out) :: dx(:)       !!!Vector of one step of the eqn set
    end subroutine step_proc
end interface
```

Figure 3. Step Abstract Type Code Listing

```
    !!!The equation (derivatives of the equation set) to be integrated
    !!  (to be stepped forward)
    !  The type for the equation set to be stepped forward should be
    !  extended from this abstract type
    type, abstract, public :: derivative_abstract_type
        !!!Parameters of the (derivatives of) equation set should be given
        !! by the extended type for that equation set
    contains
        procedure(derivative_proc), deferred :: derivative    !!!The procedure of
        !! the derivative equation to be stepped forward. This procedure takes x
        !  input and output xdot
    end type derivative_abstract_type
    !!!Interface for the deferred binding
    abstract interface
        subroutine derivative_proc(self, t, x, xdot)
            import :: derivative_abstract_type, rp
            class(derivative_abstract_type), intent(inout) :: self
            real(rp), intent(in) :: t              !!!Independent variable
            real(rp), intent(in) :: x(:)           !!!Vector of the equation set
            real(rp), intent(out) :: xdot(size(x)) !!!Vector of the derivatives of
                                                   !! equation set
        end subroutine
    end interface
```

Figure 4. Derivative Abstract Type Code Listing

```
    !!!Euler step
    type, extends(step_abstract_type), public :: step_euler_type

    contains
        procedure, nopass :: step => step_euler
    end type step_euler_type

…
…
…
    !!!Procedure for simple Euler step
    subroutine step_euler(f, t, x, h, dx)

        implicit none
        class(derivative_abstract_type) :: f
        real(rp), intent(in) :: t
        real(rp), intent(in) :: x(:)
        real(rp), intent(in) :: h
        real(rp), intent(out) :: dx(:)

        real(rp), dimension(size(x)) :: xdot

        call f%derivative(t, x, xdot)

        dx = h * xdot

    end subroutine step_euler
```

Figure 5. Euler Step Code Listing

```
    !!!RK4 step
    type, extends(step_abstract_type), public :: step_rk4_type

    contains
        procedure, nopass :: step => step_rk4
    end type step_rk4_type

…
…
…

    !!!Procedure for 4th order Runge-Kutta step
    subroutine step_rk4(f, t, x, h, dx)

        implicit none
        class(derivative_abstract_type) :: f
        real(rp), intent(in) :: t
        real(rp), intent(in) :: x(:)
        real(rp), intent(in) :: h
        real(rp), intent(out) :: dx(:)

        real(rp), dimension(size(x)) :: xdot1, xdot2, xdot3, xdot4
        real(rp) :: ho2

        ho2 = 0.5_rp * h

        call f%derivative(t, x, xdot1)           !!!1st RK4 step
        call f%derivative(t + ho2, x+ho2*xdot1, xdot2)   !!!2nd RK4 step
        call f%derivative(t + ho2, x+ho2*xdot2, xdot3)   !!!3rd RK4 step
        call f%derivative(t + h, x+h*xdot3, xdot4)       !!!4th RK4 step

        dx = h*(xdot1 + 2._rp*xdot2 + 2._rp*xdot3 + xdot4)/6._rp

    end subroutine step_rk4
```

Figure 6. RK-4 Step Code Listing

The “step” procedure needs the equation set to be solved. In order to provide a general (generic) equation set again an abstract type was used named derivative_abstract_type. Since equation set provides the derivatives, it was named with pre derivative_. By extending this class, any equation set with any parameter set can easily be implemented as given below.

Figure 7 shows the UML diagram of the class developed for the solution of the modified point mass trajectory model. It is inherited from derivative abstract type given in Figure 4. All the parameters are included in the extended type named motion_mod_point_mass_dt_type. These parameters can easily be got by means of using a “constructor”. Figure 8 shows the code listing of the class for the modified point mass trajectory model.

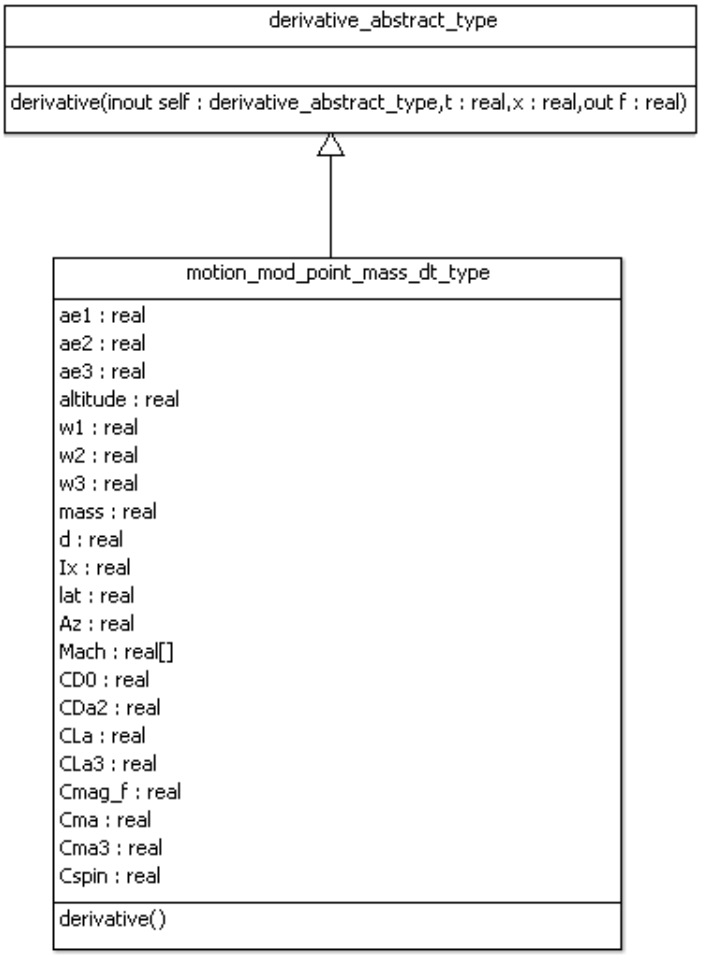


Figure 7. Modified Point Mass Equation of Motion Class UML Diagram

```
    !!!Extend derivative_abstract_type for the equation of motion according to
    !! modified point mass trajectory model
    type, extends(derivative_abstract_type) :: motion_point_mass_dt_type
        real(rp) :: ae1, ae2, ae3
        real(rp) :: altitude, w1, w2, w3
        real(rp) :: mass, d, Ix
        real(rp) :: lat, Az
        real(rp), dimension(:), allocatable :: Mach, &
                                                CD0, &
                                               CDa2, &
                                               CLa, &
                                               CLa3, &
                                              Cmag_f, &
                                                 Cma, &
                                                 Cma3, &
                                               Cspin

    contains
        procedure :: derivative => motion_point_mass_dt
    end type motion_point_mass_dt_type
…
…
…

subroutine motion_mod_point_mass_new(self, mass, d, Ix, ae1, ae2, ae3, lat, Az,&
w1, w2, w3, aerofile)

        implicit none
        class(motion_mod_point_mass_dt_type), intent(inout) :: self
        real(rp), intent(in) :: ae1, ae2, ae3, mass, d, Ix
        real(rp), intent(in) :: lat, Az
        real(rp), intent(in) :: w1, w2, w3
        character(*), intent(in) :: aerofile

        call drag_data_mod_point_mass(self%Mach, self%CD0, self%CDa2, &
                                      self%CLa, self%CLa3, self%Cmag_f, &,
                                self%CMa, &
                                      self%Cma3, self%Cspin, aerofile)
        self%ae1 = ae1
        self%ae2 = ae2
        self%ae3 = ae3

        self%w1 = w1
        self%w2 = w2
        self%w3 = w3
        self%mass = mass
        self%d = d
        self%Ix = Ix
        self%lat = lat
        self%Az = Az

    end subroutine motion_mod_point_mass_new
```

Figure 8. Code Listing of motion_mod_point_mass_dt_type Developed for the Solution of the Modified Point Mass Trajectory Model

```
    subroutine motion_mod_point_mass_dt(self, t, x, xdot)
        use :: interpolation
        implicit none
        class(motion_mod_point_mass_dt_type), intent(inout) :: self
        real(rp), intent(in) :: t
        real(rp), intent(in) :: x(:)
        real(rp), intent(out) :: xdot(size(x))
       real(rp) :: rho, P, a, Mach, d, m, lat, Az
       real(rp) :: v1, v2, v3, v, ae1, ae2, ae3, ae, Ix
       real(rp) :: CD0, CDa2, CLa, CLa3, Cmag_f, Cma, Cma3, Cspin
       real(rp), parameter :: pi = 3.14159265359_rp
       real(rp), parameter :: g0 = 9.80665_rp
       real(rp) :: g

       lat = self%lat ; Az = self%Az
       g = g0 * (1._rp - 0.0026_rp * cos(2._rp*lat))
       rho = air_density(x(6))  !!!Air density from function using standard atmosphere model
       P = air_pressure(x(6))
       a = speed_of_sound(P, rho) !!!Speed of sound from function using standard atmosphere model
       m = self%mass
       d = self%d
       Ix = self%Ix
       v1 = x(1) - self%w1
       v2 = x(2) - self%w2
       v3 = x(3) - self%w3
       v = sqrt(v1**2 + v2**2 + v3**2)
       Mach = v/a
      !!!Interpolate aerodynamic coefficients
       CD0 = interpolate(self%Mach, self%CD0, Mach)
       CDa2 = interpolate(self%Mach, self%CDa2, Mach)
       CLa = interpolate(self%Mach, self%CLa, Mach)
       CLa3 = interpolate(self%Mach, self%CLa3, Mach)
       Cmag_f = interpolate(self%Mach, self%Cmag_f, Mach)
       Cma = interpolate(self%Mach, self%Cma, Mach)
       Cma3 = interpolate(self%Mach, self%Cma3, Mach)
       Cspin = interpolate(self%Mach, self%Cspin, Mach)

        ae1 = self%ae1 ; ae2 = self%ae2 ; ae3 = self%ae3
        ae = sqrt(ae1**2 + ae2**2 + ae3**2)

       xdot(1) = .....      !!!V1dot (Equation (12))
       xdot(2) = .....      !!!V2dot (Equation (13))
       xdot(3) = .....      !!!V3dot (Equation (14))
       xdot(4) = x(1)  !!!x1dot
       xdot(5) = x(2)  !!!x2dot
       xdot(6) = x(3)  !!!x3dot
       xdot(7) = ....  !!! (Equation (16))
       xdot(8) = sqrt(x(1)**2 + x(2)**2 + x(3)**2)  !!!sdot (to obtain total path distance)

       self%ae1 = .....      !!! (Equation (17))
       self%ae2 = .....      !!! (Equation (18))
        self%ae3 = .....     !!! (Equation (19))

    end subroutine motion_mod_point_mass_dt
```

Figure 8. Code Listing of motion_mod_point_mass_dt_type Developed for the Solution of the Modified Point Mass Trajectory Model (Continued)

A main object projectile_type is generated as its UML diagram shown in Figure 9 which includes derivative_abstract_type for the modified point mass equation and step_abstract_type for the numerical solution. Code listing showing how to set the equation of motion and the integration step method as well as whole solution is shown in Figure 10. This shows the run time polymorphism in Fortran.

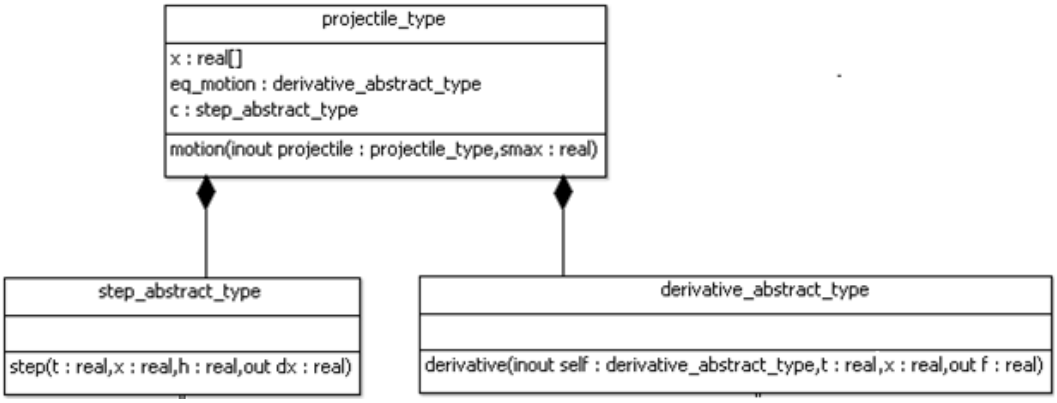


Figure 9. Projectile Class UML Diagram

```
!!!rp is real precision
!!!Get the input
…
…
…
!!!Select and set modified point mass equation of motion as equation set
allocate(projectile%eq_motion :: motion_mod_point_mass_dt_type)
!!!Set integration step method
allocate(projectile%c :: step_rk4_type)
!allocate(projectile%c :: step_euler_type)

…
…
…

        projectile%x(8) = 0._rp
        s = 0._rp    !!!Path distance
        t = 0._rp
        do while(s < smax)
            !!!Get the change of vector of equation set (dx) for the time step
            call projectile%c%step(projectile%eq_motion, t, projectile%x, dt, dx)
            !!!Integrate time step
            do i = 1, size(projectile%x)
                projectile%x(i) = projectile%x(i) + dx(i)
            enddo
            s = projectile%x(8)
            t = t + dt
        end do

…
…
```

Figure 10. Code Listing of the Solution of Projectile Motion

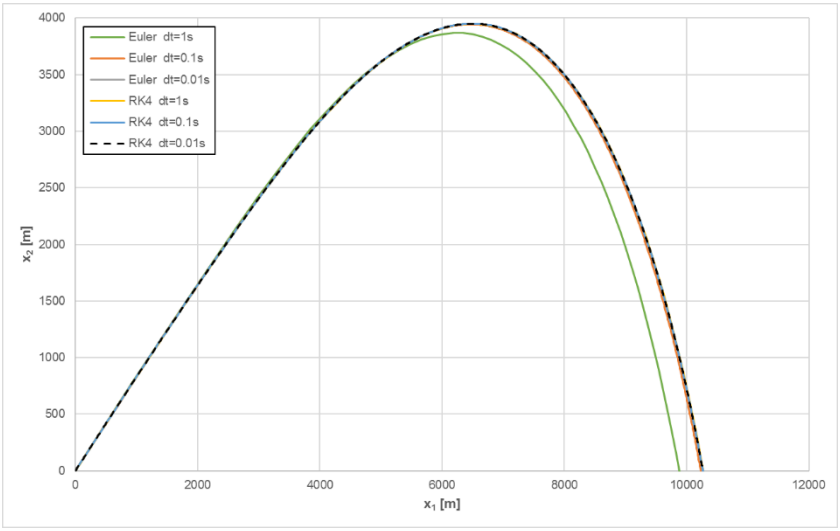
CALCULATIONS AND RESULTS

35 mm TP-T projectile trajectories were calculated using the developed code. The aerodynamic coefficients and physical properties which are given by Baranowski et.al (2020) was used. The mass, diameter and axial moment of inertia of the projectile are 0.55 kg, 0.035 and 9.7 10^{-5}, respectively.

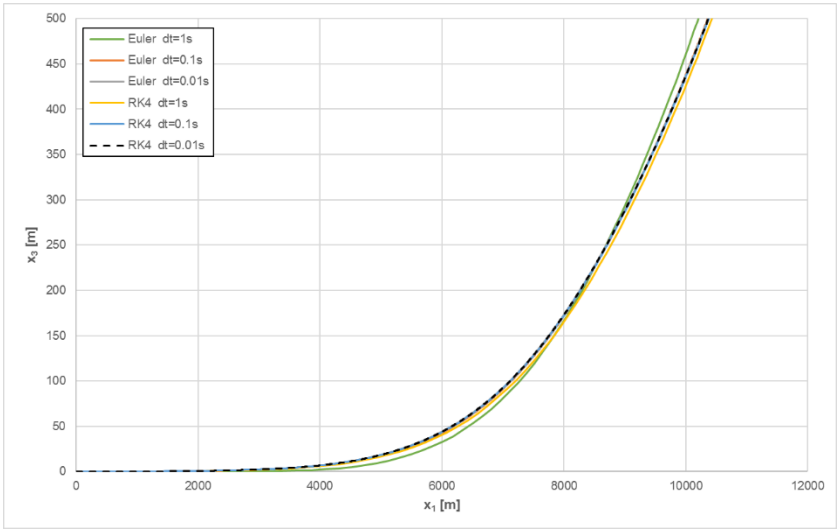
The projectile with initial conditions as the quadrant elevation QE is 710 mils and velocity equals to 1180 m/s was calculated first calculated. It was assumed to be no wind for this calculation. The trajectories on the vertical plane and horizontal plane are given in Figure 11. Calculations were performed using 1 s, 0.1 s and 0.01 s time steps and with both Euler and 4th order Runge-Kutta explicit numerical integration methods. Although it is obvious, Euler explicit

integration method results were added just to show the lose of accuracy for bigger time steps. 0.1 s and 0.01 s time steps with 4th order Runge-Kutta method provides enough accuracy. Obtained results are consistent with the results given by Baranowski et.al (2016). Elapsed times were so short that for 4th order Runge-Kutta method with 0.1 s and 0.01 s time steps were 0.016 s and 0.078 s on a laptop with i7-7700HQ @ 2.80GHz CPU.

Furthermore, the calculations were carried out for the quadrant elevations of 150 mils, 400 mils and 650 mils with 10 m/s cross wind. Results are given by Figure 12.

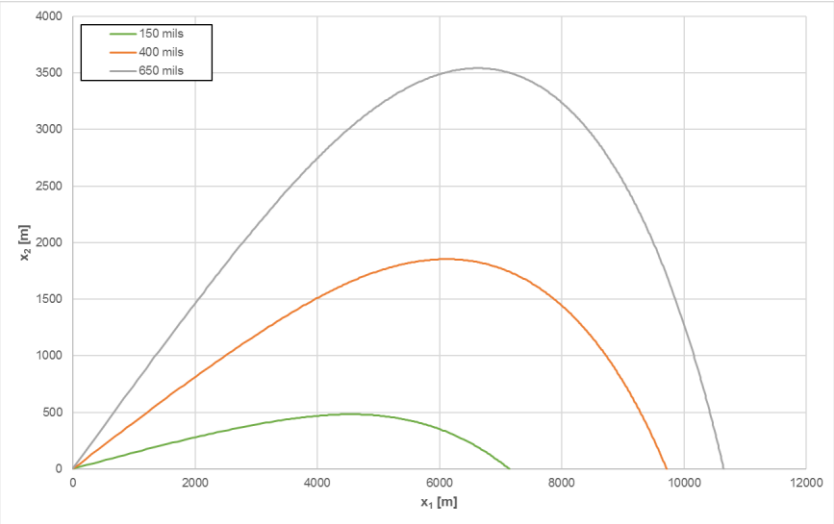


a) Vertical plane

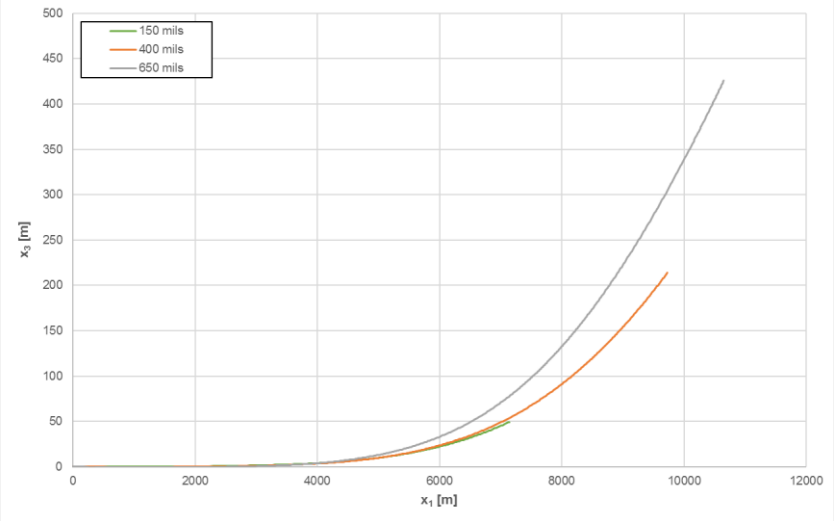


b) Horizontal plane

Figure 11. Trajectories of 35 mm TP-T Projectile for QE = 710 mils with no Wind



a) Vertical plane



b) Horizontal plane

Figure 12. Trajectories of 35 mm TP-T Projectile for QE = 150 mils/400 mils/650 mils with 10 m/s Cross Wind

CONCLUSIONS

This paper demonstrates object oriented coding technique for the solution of the modified point mass trajectory model by means of Fortran 2008 language. This method provides relatively easy code maintenance and more importantly the inheritance and polymorphism concepts ensure the further development of the code, that is to say easy additions of new methods. For example, other numerical integration methods or other equations of motion models such as 6-DOF model can easily be added using the same interfaces.

By using Fortran language, calculation results can be obtained very fast as given by this paper, which may be in the order of 100 to 500 times faster (maybe more according to the code complexity) compared to Matlab codes.

Authors of this paper strongly recommend the use of Fortran language for the computational codes and this paper shows a modern way of coding using it.

References

Baranowski, L. (2013), Feasibility Analysis of The Modified Point Mass Trajectory Model For The Need of Ground Artillery Fire Control Systems, Journal of Theoretical and Applied Mechanics 51, 3, pp. 511-522, Warsaw 2013

Baranowski, L., Gadomski, B., Szymonik, J., Majewski, P. (2016), Comparison of Explicit and Implicit Forms of the Modified Point Mass Trajectory Model, Journal of Theoretical and Applied Mechanics 54, 4, pp. 1183-1195, Warsaw 2016

Baranowski, L., Majewski, P. Szymonik, J., (2020), Explicit form of the “modified point mass trajectory model” for the use in Fire Control Systems, Bulletin of the Polish Academy of Sciences Technical Sciences, Vol. 68, No. 5, 2020

Konokman, H. E., (2021), https://github.com/konokadam/Generic-step-for-generic-equation, 2021

Lieske, R. F., Relter, M. L. (1966), Equations of Mortion for a Modified Point Mass Trajectory (1966), BRL Report No. 1314, March 1966

Metcalf, M. (1995), Abstract Data Types in Fortran 90, CERN-CN /95/1, February 1995
Simon Management Group (2006), HPC Survey Summary

Nanthaamornphong, A., Carver, J., Morris, K., & Filippone, S. (2015). Extracting uml class diagrams from object-oriented fortran: Foruml. Scientific Programming, 2015

Rouson, D., Xia, J., Xu, X. (2011), Scientific Software Design The Object-Oriented Way, Cambridge University Press

(2009), The Modified Point Mass and Five Degrees of Freedom Trajectory Models, STANAG 4355 (Ed. 3), 2009

(2021), https://www.educative.io/blog/object-oriented-programming

(2021), https://stackify.com/oop-concept-polymorphism/amp