Bulgarian Academy of Sciences. Space Research and Technology Institute.
Aerospace Research in Bulgaria. 37, 2025, Sofia

APPLYING PARALLEL SITUATIONAL ANALYSIS SOLVER TO SATELLITE – SPACE DEBRIS CLOSE APPROACH PROBLEMS. ALGORITHMS AND SUBROUTINES

Atanas Atanassov

Space Research and Technology Institute – Bulgarian Academy of Sciences
e-mail: At_M_Atanassov@yahoo.com

Keywords: *Space mission analysis and design, Situational analysis, Constraints analysis, Close approach, Conjunction analysis*

Abstract

As the number of active and passive satellites orbiting the Earth continues to increase, the likelihood of a direct collision with either another satellite or space debris also increases. Any analysis of the functionality of space systems over time must take into account the possibility of such events and the manoeuvres undertaken to mitigate the risk of a direct collision. This article considers the possibility of using the developed solver for situational analysis to solve close-approach problems between multi-satellite systems and space debris. This solver is one of several computing tools at the core of a multiphysics environment used for space mission simulations. The focus of the article is on the calculation models, algorithms, and subroutines used to develop the situational condition for "close approach" seeking.

1. Introduction

Satellite technology is developing in two main directions: the use of smaller satellites and the implementation of multi-satellite missions using smaller satellites instead of large multi-purpose ones [1, 2]. The process of space mission analysis and design involves clarifying and optimising many parameters linked to scientific instruments, satellite subsystems, orbital parameters, or templates of multi-satellite systems.

Computer simulations are playing an increasing role at all stages of preparation and operational implementation of multi-satellite missions. The development of various computational models, algorithms, and software tools is important for solving multidimensional problems related to the analysis and design of space missions, as well as for mastering the growing possibilities of the invasion of computing technologies. The application of parallel algorithms and calculations is crucial in solving large-scale problems that involve hundreds and thousands of satellites [3].

An important part of the space missions' analysis is the so-called situational analysis. The situational analysis deals with the determination of optimal time intervals suitable for the execution of satellite operations, depending on various geometric and physical conditions. This type of analysis is applied to different stages of mission preparation - starting with the conceptual study and preliminary analysis, going through mission definition, design, and development, and finishing with implementation. The close encounters between active satellites, active satellites and debris, and between debris give rise to a distinct subfield of space research that focuses on scientific problems arising from such interactions. Models and algorithms are being developed to analyse long-term and short-term changes in the space debris population [4, 5]. Various ideas are also being developed to mitigate the problems associated with the growing number of space debris [6].

The development and research of "close approach" algorithms are driven by various reasons, with the main aim being to reduce the risk of collisions between active satellites and other objects, such as passive satellites, parts of failed satellites, natural debris, or other objects. "Close encounters" are also considered in technology development to reduce the number of inactive space objects. The huge and growing number of objects presents computational problems that require solutions related to the development of parallel algorithms and the use of appropriate program models for implementation. It is also important to address the problems related to destroying mechanisms in direct collisions between satellites and other objects [7, 8].

The "close approach" problem can be parameterized by defining acceptable values for the dangerous distance between objects, time intervals for passing through the "dangerous zone", and the moment of maximum approach.

This allows the problem to be viewed as situational. This provokes us to consider the potential of applying the developed Parallel Situational Analysis Solver (PSAS) [9] to tackle such types of problems. This solver was developed at the Space Research and Technology Institute at the Bulgarian Academy of Sciences. Algorithms and some subroutines are presented in the present work.

2. Concept of situation analysis

The purpose of situational analysis is to assess the feasibility of satellite operations considering specific constraints. Each situational problem (SP) is composed of one or a conjunction of several situational conditions sc_i of various types:

$$SP = sc_1 \wedge sc_2 \wedge \ldots \wedge sc_n$$

The problem of close approach between objects in space involves determining the time interval when the objects are at a distance $D(t)$ less than a

certain threshold magnitude D_{thr} and the moment t_{close} when this distance is minimal:

$$D(t) = \sqrt{(x(t)^s - x(t)^d)^2 + (y(t)^s - y(t)^d)^2 + (z(t)^s - z(t)^d)^2} < D_{thr}$$

In the last formula, indices s and d refer to satellites and debris, respectively. The risk of collision at such a short distance is considered too likely. In this case, a specified minimum threshold of the distance between two objects, below which there is a danger of collision, is taken as a limiting condition. Taking into account the speeds with which objects move in space around the Earth, the time to pass through such a ”dangerous zone“ to a possible collision will be short. Different authors consider values of such a threshold from tens to hundreds to tens of kilometres [10, 11]. Kubasov proposes an algorithm for preventing collisions between satellites [10].

In addition to calculating the minimum distance between the two objects, the relative velocity $v(t)_{s,d}^{rel}$ and the angle θ_i between the two vectors $\vec{v}^s$ and $\vec{v}^d$ is also relevant at the time of close approach.

3. Parallel solving of ”close approach“ situational problems

In the case of solving problems of the "close approach" type, it is necessary to solve a large number of situational problems due to the increasing number of active and passive objects, especially in low Earth orbits. Applying the “perigee-apogee” filter [6] that checks possible conjunctions depending on the geometry of the objects' orbits (at some fixed time interval) eliminates the need to apply all-to-all computation procedures to an actual number of objects [12]. However, the reduced number only alleviates the problem without making it negligible.

A feature of the mathematical model for checking "close approaches" between objects results in a variable amount of calculations on different sections of their orbits (section 6). Such a model is the source of the so-called "irregular calculations" and an imbalance occurs when the calculations are parallelized [13]. This is due to the uneven distribution of computational operations between the available processors. In case of poor distribution, some processors complete their problems and are free, while others continue their calculations. The “Pool of threads” model copes with this problem. PSAS was developed to solve a variety of types and a large number of situational problems [9]. It is a processing program that consistently checks the feasibility of the conditions in a particular situational problem. The parallelization is based on computational threads organized in a variant [14] of the program model "pool of threads".

4. Situational problems editor

The situational editor, representing a dialogue form with various controls, has been developed for composing situational problems. Before creating situational

tasks, it is necessary to develop basic objects such as space missions and populations of space debris. First, a space mission is selected among those displayed in the list box control. Depending on the type of the multi-satellite system, they are selected from among possible (contextually determined) situational conditions and displayed in another list box control. The remaining objects are displayed in another window. One of them can be specified for the composition of situational problems between two multidimensional objects (a close approach between satellites and space debris, for example). Situational problems between satellites from different multi-satellite missions within a federation are possible.

Information necessary for compiling situational tasks is taken from the descriptors of the selected objects. Some of this information relates to the dimensionality of the objects (number of satellites and number of space debris). The values of addresses in the memory where the state vectors of the objects are located are also copied from their descriptors. After entering the necessary parameters and compiling situational tasks, an actual situational solver is created to solve the situational problems.

5. A situational problem description model

A descriptor of situational problems is a one-dimensional array whose elements are derived types containing the values of different attributes (parameters and constraints as well as results) of the conditions comprised in the problem. The first (zero) element of the descriptor contains control information and results about the entire situational problem. The following elements contain the values of different attributes (parameters and constraints as well as results) of the conditions in the situational problem.

Figure 1 illustrates an approach for creating situational problem descriptor objects with the means of the Fortran language. The descriptors of different situational problems are combined into a two-dimensional array with dimensions $K \times N$, where N is the number of problems and K is the maximum number of situational conditions among all problems.

The UNION statement defines groups of parameters that share memory among different situational conditions. The UNION operator specifies an area of memory used polymorphically by each function computing different situational conditions. The two parameters num_sat_61 and id_debris are located in the same memory location but are used when passing the situation condition model to two different subroutines, respectively, for checking proximity between objects from the same mission or between a satellite and space debris. The close_param array contains results from the situational analysis (Fig. 1). These results can be used in simulations at subsequent levels of the simulation model.

```
MODULE
  type     SitCond
! general parameters [cyting]
   …
  union
    map …    ! Other situational conditions
    end  map ! Other situational conditions
    map                    ! Sit_61/
      union
        map
          integer    id_debris   ! satellite - space debris & satellite – other mission satellite
        end map
        map
          integer    num_sat_61   ! satellite - satellite
        end map
      end union
        real    distance ! threshold distance
        real*8    close_param(7)        ! (1),(4)- times of entering/exiting the “dangerous zone”
                                        ! (2)- moment of maximal approach
                                        ! (3)- the magnitude of the closest distance
                                        ! (5)- the current distance between the two objects
                                        ! (6)- an angle between the objects' velocities vectors
                                        ! (7)- magnitude of the relative velocity
        real  angle_v1v2
        real  relative_velosity
    end map
    map …
    end  map ! For other situational conditions
  end union
 end type  SitCond

 type     sit_problem
  union
    map                          ! Only for solving control- contains the number of situation conditions
! general parameters [cyting]
  …
    end map
    map
      type (SitCond) sit_cond
    end map
  end union
 end type  sit_ problem
END MODULE
```

Fig. 1 Derived types for situational problems compilation

The sit_problem is a derived type that describes different situational problems [9]. The UNION statement defines groups of two MAP blocks that describe the elements of one situational problem. The first MAP block describes the zero element of the situational problem descriptor, which contains control parameters and the problem's attributes. The second MAP block allows the inheritance of the properties of each situational condition in the situational problem. The PSAS interprets situational condition attributes according to the identification code sit_code (in zero element of the situational problem). Each

function corresponding to a given situational condition interprets the attributes in a specific way (kind of polymorphism) depending on a corresponding MAP block.

6. Interpolation of the state vectors

Note that it is impossible to solve a problem of "close approach" between two objects based on state vectors determined by numerical integration with a time step Δt in the order of tens of seconds. This is because at velocities of the order of several kilometres per second at which space objects travel (depending on the altitudes at the points on the orbit), the distances travelled per integration step can be of the order of hundreds of kilometres. Thus, exact values for the moment of minimum distance and moments of entry and exit from the "dangerous zone" cannot be calculated. It is necessary to use interpolation to find the coordinates of the bodies at any time within the step. The Lagrangian method [15] is used in the present work. The coordinates and velocities of the objects are calculated with a constant time step by applying the Parallel Integrator of systems of differential equations for space objects motion integration [14]. To apply an interpolation method, the state vectors' last few values must be available. At the stage of creating the model of a satellite mission (or other main object, for example, space debris), the following structure is created:

```
type      Nodes_Lagrange
 integer    nodes,count_nodes
 integer    denominator_nodes_adr
 integer    t_nodes_adr,
 integer  xv_nodes_adr
end type  Nodes_Lagrange
```

The attributes of this structure are assigned values when the space mission model is created, and its address is written to the mission descriptor. Access to this descriptor is provided when creating situational task models. The address of the mentioned structure, along with the addresses of the state vectors of the objects, the Lagrange coefficients, and the number of interpolation nodes, is taken from this descriptor and passed to the situation solver for solving situational problems. This is done at the creation stage of the mission model after the thread pool is created.

7. Algorithm and subroutine for checking for situational condition

The "dangerous zone" search algorithm is based on analyzing the distance function between objects $D(t)$. This function is unimodal over a fairly wide time interval. Therefore, within one integration step Δt, the behaviour of the derivative of the distance function at both ends of the step ($D'(t)$ and $D'(t+dt)$) allows to establish the presence of a minimum. At the heart of the algorithm is the construction a1: IF-THEN-ELSE IF-END IF a1 (Appendix A). If the derivative at

both ends of the step dt is negative, the distance decreases. The condition (derivative_1.LT..0D0 .AND. derivative_2.LT..0D0) is checked. In this case, the threshold of dangerous convergence between the two objects is checked. This requires that the threshold value be intermediate to the distance values at either end of the time step. For this purpose, the construction b1: IF-THEN-ELSE IF-END IF b1 is used. The (Dist_n .GT. distance .AND. Dist_k .GT. distance) condition check applies outside the “dangerous zone” whose dimensions are determined by the value of the parameter distance. Similarly, the condition (Dist_n .LT. distance .AND. Dist_k .LT. distance) checks for falling "inside" the “dangerous zone”. When the condition (Dist_n .GT. distance .AND. Dist_k .LT. distance) is fulfilled, the moment of entry of the satellite t_dist into the “dangerous zone” is determined. This is done with the Find_threshold() subroutine (Appendix A).

Analogously, with the condition (derivative_1.GT..0D0 .AND. derivative_2.GT..0D0), an increase in the distance between the two objects is checked. In this case, the construct b1: IF-THEN-ELSE IF-END IF b1 aims to detect the end of the “dangerous zone” and to determine again with the subroutine Find_treshold() the moment of exit from it. This subroutine is the result of a combination of two separate subroutines for determining the times of entry or exit from the “dangerous zone”. Allowed values of the first actual parameter var are 1 or 2, for the beginning and exit times, respectively.

The condition (derivative_1.GE..0D0 .AND. derivative_2.LE..0D0) corresponds to reaching a local maximum of the function $D(t)$ and is only for completeness, in fact, its check can be removed. The last condition (derivative_1.LE..0D0.AND.derivative_2.GE..0D0) refers to finding a local minimum of the function $D(t)$ contained within the current step. This is accomplished with the Zero_deriv() subroutine, which looks for the instant in time for which the derivative $D'(t)$ has a zero value.

It is possible to pass through the danger zone in a single integration time step, depending on the relative velocities of the objects and the step size Δt. For this purpose, after determining the values of the minimum distance and the moment when it is reached, it is also checked for the moments of entry and exit from the “dangerous zone”. It is possible for one or both of the moments to fall within the time frame of the current step Δt.

8. Auxilary subroutines

A subroutine that calculates the distance between two objects

The subroutine Lagrange_interp calculates the value of the function $D(t)$ at any time $t \in [t_{k-1}, t_k]$, where k is the serial number of the node in the Lagrange interpolation. Partially pre-calculated Lagrange coefficients are used. With additional calculations using the coordinates of the objects in the last few moments,

the final coefficients are calculated. After determining the coordinates of the objects at the desired moment, the distance between them is calculated.

Subroutine to determine the derivative of the function D(t)

The subroutine First_deriv determines the derivative $D'(t)$ of the distance function. This is done by determining the values of the distance function at two close points in time $D(t)$ and $D(t-\Delta t)$. Finally, the value of the derivative

$$D'(t) = (D(t) - D(t-\Delta t))/\Delta t.$$

Subroutine for finding the boundaries of the "dangerous zone"

The subroutine Find_threshold determines the boundaries of the "dangerous zone" at intermediate times $t_{beg/end} \in [t_{k-1}, t_k]$. It combines two separate subroutines, depending on the var parameter, determining the moments when the objects are at the beginning or the end of the close approach area.

The search is done with an iterative method, halving each subsequent interval and checking the distance value (defined by the program) if it is greater or less than the set limit value. Depending on the results of the checks, the limits of the interval change. The process is always convergent.

Subroutine for finding minimum distance

The subroutine searches for the minimum of the function $D(t)$ in the interval for which the derivative is negative at the beginning and positive at the end. The function $D(t)$ is unimodal and has a single minimum within the considered interval. Again, the bisection method is applied, using the First_deriv subroutine to determine the derivative in the middle of the interval and check its value. Depending on the sign of the derivative at the midpoint, one of the boundaries of the considered interval changes. Thus, the interval is shortened with each iteration. The process is convergent and terminates when a value for the derivative has reached some small limiting value.

We should note that an interface to connect objects from different missions was developed to solve inter-mission situational problems [16]. In addition to solving situational tasks, this approach is also applicable in other cases when information from different missions is necessary.

9. Examples of applying the presented approach and subroutines

Seven orbits have been selected for testing the approach and subroutines that determine the moments of passing through the "dangerous zone". They have equal semi-major axes a_n = 7,200 km, eccentricities e_n = .001, and perigee arguments ω_n = .0 deg. Different values of inclinations i_n and ascending nodes Ω_n, given in Table 1, are selected to ensure an approach between the objects.

Table 1. Parameters of the orbits used in the numerical experiments

Orbits	1	2	3	4	5	6	7
Inclination [deg]	.0	1.0	90.0	180.0	90.0	45.0	135.0
Ascending node [deg]	0.0	0.0	0.0	0.0	90.0	0.0	90.0

Within the performed experiments, the integration of the equations of motion was performed with different time steps. The test results at an integration step $\Delta t = 10$ s are presented in Table 2. The accuracy of the calculated times depends only on the integration accuracy!

Table 2. Calculation results based on the orbits from Table 1

Pairs of orbits	Minimum distance [m]	Entering the "dangerous zone" [s]	Moment of minimum distance [s]	Exiting the "dangerous zone [s]	Relative velocity [m/s]
1-2	.00050	2963.04955	3040.21874	3117.38792	129.72
1-3	.04050	3039.26739	3040.21874	3041.17010	10511.158
1-4	.05727	3039.54603	3040.21874	3040.89145	14865.022
3-5 I	.02433	1517.15041	1518.10078	1519.05115	10522.077
3-5 II	.03643	4561.38629	4562.33665	4563.28702	10522.078
6-7 I	.06134	922.07540	922.85091	923.62641	12894.581
6-7 II	.02911	3965.57317	3966.34961	3967.12604	12879.140

When the orbital planes make a small angle (the pair of objects 1-2) the "dangerous zone" is relatively extended in time. At large angles, however, passing through the "dangerous zone" occurs in shorter time intervals.

10. Conclusion and outlook

An algorithm and subroutines for checking a situational condition to search for a close approach between two orbital objects are presented in this article. These subroutines are designed to function within the framework of the situation solver being developed. This class of situational tasks can be linked with others, such as the destruction of space debris and the destruction of whole satellites, subsystems, or individual devices. Furthermore, the time functionality of satellites within individual missions (or within federations) can be explored to perform the targeted tasks.

Appendix A. Source code of the subroutines checking the situational conditions

```
!*****************************************************************************************
! <Sit___61>- Close approach between "satellite-satellite"
!
!   node_t(node,1)= t
!   object1_nodes   - coordinates of the first satellite
```

```
!   object2_nodes   - coordinates of the second satellite
!   <First_deriv>    - calculates the derivative at the moment t
!   <First_treshold> - determines the first moment t when the distance is equal to <distance>
!         <Flag_1> - <First_treshold> was passed
!   <Seconf_treshold>- determines the last moment t when the distance is equal to <distance>
!         <Flag_2> - <Seconf_treshold> was passed
!   <close_param>    - 7 elements array;
!   close_param(1)  - passing moments for <First_treshold>
!   close_param(2)  - passing moments for <Dist_close>
!   close_param(3)  - magnitude of <Dist_close>
!   close_param(4)  - passing moments for <Second_treshold>
!   close_param(5)  - the current distance between the two objects
!   close_param(6)  - an angle between the objects' velocities vectors
!   close_param(7)  - magnitude of the relative velocity
!.....*
FUNCTION     Sit___61(t,dt,object1_adr,object2_adr,nodes,nodes_count,node_t_adr,adr_znam_nodes, &
                          distance,close_param,angle_v1v2, fl_rezults,duration,begin_sit,dt_sit,t12,id_debris)
  logical    Sit___61,                fl_rezults,       begin_sit
  integer              object1_adr,object2_adr, adr_znam_nodes
  real                      distance,angle_v1v2,duration,            t12*8(2,3)
  real*8           t,dt, tt,close_param(5)
  real*8              object1_nodes(6,nodes),object2_nodes(6,nodes),node_t(nodes),node_znam(nodes)
  logical      flag,flag_1[save],flag_2[save]
  real*8     derivative_1,derivative_2,Dist,t_dist,Dist_n,Dist_k,t_close,dist_close
  real*8     v1v2_delta,delta_v1v2
!_____________________________________
  AUTOMATIC     flag,flag_1,flag_2,Dist_n,Dist_k,derivative_1,derivative_2,Dist_close,t_dist
  POINTER(node_t_adr,node_t);            POINTER(adr_znam_nodes,node_znam)
  POINTER(object1_adr,object1_nodes); POINTER(object2_adr,object2_nodes)

  IF(nodes_count.LT.nodes) THEN
          Sit___61=.false.;      RETURN
    ENDIF

              t= node_t(nodes - 1);
        Dist_n=.0D0; Dist_k=.0D0
    DO          i=1,3
        Dist_n= DIST_n + (object1_nodes(i,nodes-1) - object2_nodes(i,nodes-1))**2
        Dist_k= DIST_k + (object1_nodes(i,nodes  ) - object2_nodes(i,nodes  ))**2
   END DO; Dist_n= SQRT(Dist_n); Dist_k= SQRT(Dist_k); close_param(5)= Dist_k

          derivative_1= First_deriv (node_t(nodes-1),nodes,node_t,object1_nodes,object2_nodes,node_znam)
          derivative_2= First_deriv (node_t(nodes  ),nodes,node_t,object1_nodes,object2_nodes,node_znam)
a1:    IF(derivative_1.LT..0D0.AND.derivative_2.LT..0D0) THEN ! Decreasing distance
            flag_2=.false.
  b1:         IF(Dist_n  .GT. distance .AND. Dist_k .GT. distance) THEN ! There is no threshold
        ELSEIF(Dist_n  .LT. distance .AND. Dist_k  .LT. distance) THEN ! The threshold is passed
        ELSEIF(Dist_n .GT. distance  .AND. Dist_k .LT. distance) THEN ! Passing the threshold
                  flag_1= Find_treshold(1, node_t,distance, t_dist, Dist_n,Dist_k,nodes,node_znam, &
                                                                          object1_nodes,object2_nodes)
                                          close_param(1)= t_dist;       flag_2=.true.
         ENDIF b1
 ELSEIF(derivative_1.GT..0D0.AND.derivative_2.GT..0D0) THEN ! Increasing distance
            flag_1=.false.
  b2:         IF(Dist_n .GT. distance .AND. Dist_k .GT. distance) THEN ! There is no threshold
        ELSEIF(Dist_n .LT. distance .AND. DIst_k .LT. distance) THEN ! The threshold is not passed yet
        ELSEIF(Dist_n .LT. distance .AND. Dist_k .GT. distance) THEN ! Ima prag
                  flag_2= Second_treshold(node_t,distance,t_dist,Dist_n,Dist_k,nodes,node_znam, &
```

```
                                                    object1_nodes,object2_nodes)
                                    close_param(4)= t_dist;        flag_2=.true.
          ENDIF b2
  ELSEIF(derivative_1.GE..0D0.AND.derivative_2.LE..0D0) THEN ! Local maximum
  ELSEIF(derivative_1.LE..0D0.AND.derivative_2.GE..0D0) THEN ! Local minimum
          Dist_close= Zero_deriv(node_t,nodes,t_close,object1_nodes,object2_nodes,node_znam);
                          close_param(2)= t_close
                          close_param(3)= Dist_close; close_param(5)= Dist_close

       delta_v1v2= Velosity_interp(t_close,nodes,node_t,object1_nodes,object2_nodes,node_znam);

   b3: IF(Dist_n.GT.distance.AND.Dist_close.LT.distance) THEN ! If the first threshold isn't passed
          flag_1= First_treshol (node_t(nodes-1),t_close,node_t,distance,t_dist,Dist_n,Dist_k,nodes, &
                                                  node_znam,object1_nodes, object2_nodes)
                                                close_param(1)= t_dist;        flag_1=.true.
       ENDIF b3

   b4: IF(Dist_close.LT.distance.AND.Dist_k.GT.distance) THEN ! Ima prag v ostavashtata chast na stapkata
          flag_2= Second_treshol (t_close,node_t(nodes),node_t,distance,t_dist,Dist_n,Dist_k,nodes, &
                                                    node_znam,object1_nodes,object2_nodes)
                                                  close_param(4)= t_dist;        flag_2=.true.
        ENDIF b4
  ENDIF a1;
                    Sit___61=.true.
END FUNCTION  Sit___61

!**********************************************************************
! <First_deriv>- determines the first derivative in the moment <t>
!.....*
FUNCTION   First_deriv(t,nodes,node_t,      node_1,          node_2,          node_znam)
  real*8        First_deriv
  real*8            t,node_t(nodes),          node_1(6,nodes),node_2(6,nodes),node_znam(nodes)
  real*8        dt/.00001/, Dist1, Dist0, Lagrange_interp
   AUTOMATIC  Dist1,Dist0

      Dist1= Lagrange_interp (t   ,nodes, node_t,node_1,node_2,node_znam)
      Dist0= Lagrange_interp (t-dt,nodes, node_t,node_1,node_2,node_znam);

                    First_deriv= (Dist1 - Dist0)/dt
END FUNCTION  First_deriv

!**********************************************************************
! First derivative - zero; minimum distance
!_____________________________________________________________________
FUNCTION  Zero_deriv(node_t,nodes,  t_close, node_1,          node_2,          node_znam)
  real*8        Zero_deriv
  real*8                  node_t(nodes),t_close, node_1(6,nodes),node_2(6,nodes),node_znam(nodes)
  real*8    First_deriv, Lagrange_interp, t_1,t_2,tm, Dist_close
  AUTOMATIC     t_1,t_2,tm,Dist_close

                        t_1= node_t(nodes-1);   t_2= node_t(nodes);   tm= t_1 + .5*(t_2 - t_1)
                    derivative= First_deriv(tm,nodes,node_t,node_1,node_2,node_znam);
  DO WHILE(ABS(derivative).GT..00001.AND.(t_2-t_1).GE..00001);
                  IF(derivative.GT..0D0) THEN
                        t_2= tm
            ELSEIF(derivative.LT..0D0) THEN
                        t_1= tm
             ENDIF; tm= t_1 + .5*(t_2 - t_1)
```

```
                    derivative= First_deriv(tm,nodes,node_t,node_1,node_2,node_znam);
  END DO;
                                Dist_close= Lagrange_interp (tm,nodes, node_t,node_1,node_2,node_znam);
                    Zero_deriv= Dist_close; t_close= tm
END FUNCTION    Zero_deriv

!*********************************************************
! <Find_treshold>- finds the start and end times of the "dangerous zone"
!        <var>- var=1 - finds the first threshold
!               var=2 - finds the second threshold
!_________________________________________________________
FUNCTION  Find_treshold(var,tn,tk,node_t,distance,ts,Dist_n,Dist_k,nodes,node_znam,   node_1,     node_2)
  logical       Find_treshold
  integer                   var
  real                                  distance
  real*8                        tn,tk,node_t(nodes),  ts,Dist_n,Dist_k,        node_znam(1),node_1(1),node_2(1)
  real*8    Disr_n,Disr_k,Dist_t,tt,t1,t2,Lagrange_interp,delta
  real*8,parameter ::  tol=.01
  AUTOMATIC  Disr_n,Disr_k,Dist_t,t1,t2,tt

          t1= tn; t2= tk; Disr_n= Dist_n; Disr_k= Dist_k; Find_treshold=.false.;!var_12= 2
          tt= t1 + .5*(t2 - t1)
      IF(var.EQ.1) THEN; delta= Disr_n - Disr_k
                   ELSE; delta= Disr_k - Disr_n
        ENDIF
a:  DO WHILE(delta.GT.tol)
        Dist_t= Lagrange_interp(tt,nodes,node_t,node_1,node_2,node_znam)
          IF(Dist_t.GT.distance) THEN
            IF(var.EQ.1) THEN; Disr_n= Dist_t; t1= tt
                         ELSE; Disr_k= Dist_t; t2= tt
              ENDIF
      ELSEIF(Dist_t.LE.distance) THEN
            IF(var.EQ.1) THEN; Disr_k= Dist_t; t2= tt
                         ELSE; Disr_n= Dist_t; t1= tt
              ENDIF
       ENDIF;
         ts= tt; tt= t1 + .5*(t2 - t1)
            IF(var.EQ.1) THEN; delta= Disr_n - Disr_k
                         ELSE; delta= Disr_k - Disr_n
              ENDIF
   END DO a;
                Find_treshold=.true.
END FUNCTION    Find_treshold

!*****************************************************************************************
! <Lagrange_interp>- determines the minimal distance between two objects in a time moment <t>
!                                                       Lagrange interpolation is applied
!_____________________________________________________________________________________________
FUNCTION  Lagrange_interp(t,nodes,node_t,        node_s,        node_d,        node_znam)
  real*8      Lagrange_interp
  real*8                    t,      node_t(nodes),node_s(6,nodes),node_d(6,nodes),node_znam(nodes)
  real*8      rt, numerator, Sx, Sy, Sz, Dx, Dy, Dz, coeff_Lagrange
  AUTOMATIC  numerator, Sx, Sy, Sz, Dx, Dy, Dz, coeff_Lagrange

          Sx= 0.D0; Sy= 0.D0; Sz= 0.D0; Dx= 0.D0; Dy= 0.D0; Dz= 0.D0; rt= t;
   DO     nd=1,nodes
      numerator = 1.D0;
     DO   md=1,nodes
       IF(md.NE.nd) THEN
```

```
                numerator = numerator *(rt - node_t(md))
            ENDIF
        END DO;
          coeff_Lagrange= numerator *node_znam(nd);
                        Sx= Sx + node_s(1,nd)*coeff_Lagrange;    Dx= Dx + node_d(1,nd)*coeff_Lagrange
                        Sy= Sy + node_s(2,nd)*coeff_Lagrange;    Dy= Dy + node_d(2,nd)*coeff_Lagrange
                        Sz= Sz + node_s(3,nd)*coeff_Lagrange;    Dz= Dz + node_d(3,nd)*coeff_Lagrange
    END DO;
                    Lagrange_interp= SQRT((Sx - Dx)**2 + (Sy - Dy)**2 + (Sz - Dz)**2);
END FUNCTION  Lagrange_interp
```

References

1. Poghosyan, A., Lluch, I., Matevosyan, H., Lamb, A., Moreno, C. A., et al., (2016), Unified classification for distributed satellite systems. In *4th International Federated and Fractionated Satellite Systems Workshop, 10-11 Oct, Rome, Italy*.
2. Selva, D., A., Golkar, O. Korobova, I. L. I. Cruz, P. Collopy, O.L. de Weck, (2017), Distributed earth satellite systems: What is needed to move forward?. *Journal of Aerospace Information Systems*, *14*(8), pp. 412–438.
3. Nelson, B., Yang Yang, F., Carlino, R., Dono Perez, A., Faber, N., Henze, C., ... & Stupl, J., (2015). Implementation of an Open-Scenario, Long-Term Space Debris Simulation Approach. In *AMOS 2015 Advanced Maui Optical and Space Surveillance Conference* (No. ARC-E-DAA-TN26775).
4. Klinkrad, H., Wegener, P., Wiedemann, C., Bendisch, J. and Krag, H., (2006), Modeling of the current space debris environment. *Space Debris: Models and Risk Analysis*, pp. 59–114.
5. Назаренко, А. И. (2013), Моделирование космического мусора. *М.: ИКИ РАН*.
6. Casanova, D., Tardioli, C. and Lemaitre, A., (2014), Space debris collision avoidance using a three-filter sequence. *Monthly Notices of the Royal Astronomical Society*, *442*(4), pp. 3235–3242.
7. Киселев, А.Б. and Яруничев, В.А., (2009), К исследованию фрагментации частиц космического мусора при высокоскоростном соударении. *Вестник Московского университета. Серия 1. Математика. Механика*, (2), pp. 26–35.
8. Smirnov, N. N., Kiselev, A. B., Kondratyev, K. A. and Zolkin, S. N., (2010), Impact of debris particles on space structures modeling. *Acta Astronautica*, *67*(3-4), pp. 333–343.
9. Atanassov, A.M., (2016), Parallel satellite orbital situational problems solver for space missions design and control, *Adv. Space Res.*, v. 58, 9, 2016, pp. 1819–1826.
10. Кубасов, И. Ю., Хасанов, А. Ю. and Блюдов, Е. С., (2022), АЛГОРИТМ ПРЕДОТВРАЩЕНИЯ СТОЛКНОВЕНИЯ КОСМИЧЕСКИХ АППАРАТОВ. *Известия Тульского государственного университета. Технические науки*, (8), pp. 144–149.
11. Hall, R., Alfano, S., & Ocampo, A., (2010), Advances in satellite conjunction analysis. In *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*.
12. https://www.esa.int/Space_Safety/Space_Debris/Space_debris_by_the_numbers
13. Rauber, T., Rünger G., (2010), Parallel Programming: For Multicore and Cluster Systems. Springer. 455 p; DOI 10.1007/978-3-642-04818-0

14. Кальницкий, Л.А., Добротин, Д.А., Жевержеев, В.Ф., and Сапогов, Н.А., (1976), Специальный курс высшей математики для втузов.
15. Atanassov, A.M., (2014), Parallel, adaptive, multi-object trajectory integrator for space simulation applications. *Advances in Space Research*, *54*(8), pp.1581-1589.
16. Atanassov, A.M., (2023), APPLYING SITUATION ANALYSIS SOLVER TO SATELLITE-SPACE DEBRIS CLOSE APPROACHES PROBLEMS. INTERFACE BETWEEN MODELS., in Proceedings of the Nineteenth International Scientific conference SES2023, 24-26.10.2023, Sofia, Bulgaria, pp. 94–99. http://space.bas.bg/SES/archive/SES%202023_DOKLADI/1_Space%20Physics/11_Atanassov.pdf

ПРИМЕНЕНИЕ ПАРАЛЛЕЛЬНОГО РЕШАТЕЛЯ ДЛЯ СИТУАЦИОННОГО АНАЛИЗА В ЗАДАЧАХ СБЛИЖЕНИЯ ИСКУССТВЕННЫХ СПУТНИКОВ С ДРУГИМИ СПУТНИКАМИ И С КОСМИЧЕСКИМ МУСОРОМ. АЛГОРИТМЫ И ПОДПРОГРАММЫ

Атанас Атанасов

Аннотация

По мере увеличения количества активных и пассивных искусственных спутников Земли возрастает вероятность прямых столкновений, как между различными спутниками, так и между спутниками и космическим мусором. Анализ функционирования космических систем во времени должен учитывать возможность таких событий, а также маневры, выполняемые для снижения опасности прямого столкновения. Рассмотрена возможность использования разработанного решателя ситуационного анализа для решения задач сбли-жения многоспутниковых систем и космического мусора. Этот решатель является одним из вычислительных средств, лежащих в основе меж-дисциплинарной компьютерной среды для моделирования космических миссий. Основное внимание уделяется описанию вычислительных алгоритмов и подпрограмм.