МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

О.Н. Наумов

Практикум по динамике и управлению движением космических тросовых систем.

Часть вторая.

Электронное учебное пособие

Работа выполнена по мероприятию блока 2 «Развитие и повышение эффективности научно-инновационной деятельности» и блока 3 «Развитие информационной научно-образовательной среды и инфраструктуры» Программы развития СГАУ на 2009 – 2018 годы по проекту «Разработка учебно-исследовательского комплекса автоматизации моделирования алгоритмов параллельных вычислений» Соглашение № 2/8... от 03 июня 2013 г.

САМАРА
2013

УДК 629.78(075)
ББК 39.62я7
Н 342

Авторы: **Наумов Олег Николаевич**

Рецензенты:
Заместитель директора ИПУСС РАН по научной работе, д.т.н. Ю.Н.Горелов
Профессор кафедры космического машиностроения СГАУ, д.т.н. С.А.Ишков

Компьютерная верстка: Наумов О.Н.

Наумов, О.Н. Практикум по динамике и управлению движением космических тросовых систем [Электронный ресурс].Ч**2:** электрон. учеб. пособие / О.Н.Наумов; М-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С.П. Королева (нац. исслед. ун-т). – Электрон. текстовые и граф. дан. (1,6 Мбайт). – Самара, 2013. – 1 эл. опт. диск (CD-ROM).

В учебном пособии описываются особенности моделирования динамики космической тросовой системы в среде **PGraph.**

Пособие предназначено для магистров по направлению 010300.68 «Фундаментальная информатика и информационные технологии» (ФГОС-3), изучающих дисциплину «Современные методы разработки программных комплексов» в 9 семестре. Учебное пособие может быть полезно для подготовки выпускных работ бакалавров и магистров по указанному направлению. Разработано на кафедре программных систем.

Содержание

1. Применение космических тросовых систем

Космические тросовые системы (КТС) представляют собой совокупность двух и более твердых тел связанных тросами. Они могут быть использованы для решения различных практических задач в космосе: доставки малых капсул с орбиты на поверхность Земли, исследования гравитационного и магнитного поля планет и их спутников, создание искусственной гравитации во время дальних космических миссий и т. д. За последние полвека было проведено значительное количество натурных экспериментов в космосе с использованием КТС, наиболее известные из них: TSS-1, TSS-1R, OEDIPUS-A, OEDIPUS-C, SEDS-1, SEDS-2, YES2 [1-12].

Первые эксперименты по развёртыванию тросовых систем были проведены в 60-е года XX века. В 1966 г. во время полетов кораблей «Джемини-11» и «Джемини-12» были проведены эксперименты по движению кораблей в связке со ступенью «Аджена». Длина синтетического троса составляла при этом 30 м. Связка с «Джемини-11» была приведена во вращательное движение, а связка с «Джемини-12» была приведена в состояние гравитационной стабилизации [5].

В дальнейшем были проведены эксперименты по развёртыванию тросовых систем с различным успехом и с различными целями. Наиболее существенными были проекты SEDS-1 и SEDS-2. Эксперимент SEDS-1 был проведен в марте 1993 года и его основной целью было доказать возможность того, что можно развернуть трос на длину 20 километров, и в дальнейшем использовать данный механизм для доставки полезной нагрузки на более низкие орбиты. SEDS-2 был проведен в марте 1994 года, в отличие от своего предшественника данный проект использовал возможность управления развёртыванием троса по принципу закрытого цикла (обратной связи), идея была в том что шаговый мотор управлял количеством витков троса на специальной катушке и тем самым можно было изменять скорость

развёртывания троса за счет изменения величины управляющей силы, которая зависела от числа витков троса на катушке. Основной целью эксперимента было развернуть трос на 20 километров и плавно стабилизировать тросовую систему вдоль местной геовертикали. Одной из особенностей эксперимента была необходимость проверить долгосрочные эволюционные эффекты в динамике КТС, но из-за столкновения с микрометеоритом либо с космическим мусором, трос оборвался, проработав, таким образом, пять дней [6].

В начале 70-х годов NASA совместно с Итальянским Космическим Агентством поставила задачу о разработке возможности развёртывания космических тросов большой длины, и данный проект впоследствии был назван TSS-1. Идея данного эксперимента заключалась в развёртывании тросовой системы с борта космического корабля Шаттл. Развернувшийся трос должен был бы использоваться для исследования физических свойств разряженной плазмы окружающего космического пространства. Реализация эксперимента TSS-1 было начато 31 июля 1992 года, но в результате трос был развёрнут только на 268 метров, тем самым эксперимент был признан не удачным. Эксперимент TSS-1 был дополнен новым экспериментом TSS-1R, реализация которого началась 22 февраля 1996 года. Цель эксперимента состояла в развёртывании электродинамической КТС. Однако эксперимент также не удался: вместо запланированных 20,7 километров, трос развернулся только на 19,7 километра [6].

Для изучения ионосферы в 1989 году был проведен канадский эксперимент OEDIPUS-A. Он заключался в том, что проводилось развёртывания троса с космическим зондом. В результате развёртывания была сформирована космическая структура, состоящая из двух концевых тел с массами 84 и 131 кг, и с тросом длиной 958 м. Основная цель данного проекта заключалась в том, чтобы провести исследования распространения плоских высокочастотных волн в окружающей зонд плазме. Тросовая система была сориентирована вдоль магнитного поля Земли.

Проект OEDIPUS-C, осуществленный в 1995 году, в основном был похож на своего предшественника OEDIPUS-A, но имел ряд важных нововведений: зонд был выведен на более высокую траекторию с апогеем 843 километра, и трос развернулся на большую длину – 1174 метра [6]. Более высокая орбита имела преимущества, так как характеризовалась более высокой плотностью плазмы, и тем самым лучше позволяла исследовать характеристики взаимодействия проводящего троса и окружающей его среды. Данный эксперимент имел также целью исследовать динамику тросовой системы, использующей проводящий трос, что его также отличает от проекта OEDIPUS-A. Схема проекта изображена на рисунке 1.

Рис. 1 – Эксперимент OEDIPUS- C [8]

Отличительной чертой проектов серии OEDIPUS было то, что концевые тела имели массы одного порядка.

В начале 90-х годов XX века началась работа под руководством голландской фирмы Delta-Utec и Европейского Космического Агентства (ЕКА) над проектом (Young engineer satellite – спутник молодых инженеров) YES, который в начале 2000-х годов перерос в проект YES2.

В данном проекте участвовало большое количество студентов из различных стран, как из Европы, так и из Канады, Японии, России. В основе проекта лежала идея развертывания космической тросовой системы для доставки полезной нагрузки на Землю с борта базового космического аппарата (так называемая космическая почта). В результате развёртывания троса полезная нагрузка опускается на более низкую орбиту и далее трос обрезается как со стороны КА, так и со стороны спускаемой капсулы. Далее происходит свободный спуск спускаемой капсулы с полезной нагрузкой.

Данный проект был реализован в 2007 году. Космический трос был развернут с борта КА «Фотон М-3».

В результате тросовая система развернулась на величину порядка 32 километров, что превысило запланированную длину 30 километров, данная ошибка по результатам исследований послеполетных данных была вызвана сбоем в работе электронной аппаратуры. Развёртывание троса должно было проходить по заранее заданному закону, контролируемыми параметрами которого были длина и скорость троса, в итоге при проведении эксперимента в какой-то момент не удалось затормозить трос, и он развернулся на всю возможную длину, и далее произошло обрезание троса в заданный момент времени. Вследствие данного события, капсула «Фотино» вошла в верхние слои атмосферы, сильно отклонившись от расчетной траектории (капсула не найдена до сих пор) [9 – 11]. Схема спуска показана на рис. 2 [12].

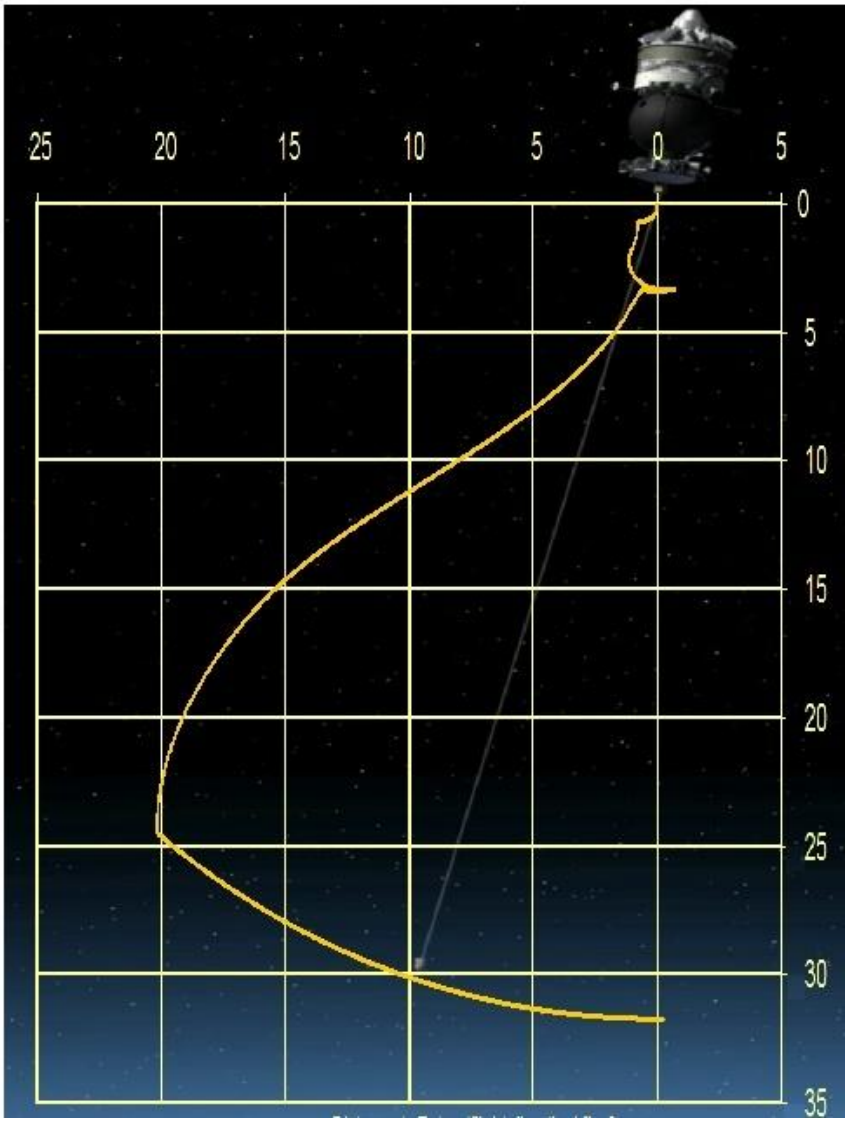


Рис. 2 – Траектория спускаемой капсулы в проекте YES2

На рис. 2 по осям абсцисс и ординат отложена величина отклонения троса от точки подвеса в километрах. Из рисунка видно, что процесс развёртывания состоял из двух этапов первый этап: развёртывание троса на величину порядка 3.6 километра (этап медленного развёртывания), второй этап: развёртывание на величину порядка 32 километра (этап быстрого развёртывания). Каждый этап заканчивается маятниковым движением и выходом троса со спускаемой капсулой «Фотино» на положение местной геовертикали.

Помимо рассмотренных выше космических проектов, которые были уже реализованы, в последнее время появилось большое число гипотетических проектов связанных с использованием космических тросов,

наиболее интересные из них это проекты по использованию КТС для осуществления долгосрочных космических перелетов. В их основе лежит идея двух вращающихся КТС, одна из которых располагается на Земной орбите, другая на Лунной или Марсианской. Данная схема позволяет перевести полезную нагрузку с эллиптической земной орбиты за счет использования Земной КТС на переходную орбиту к Марсу или Луне, где в самом конце своей переходной траектории полезная нагрузка будет захвачена второй КТС, которая в свою очередь движется по эллиптической орбите вокруг Марса или Луны. Данная схема позволяет осуществлять перемещения полезной нагрузки туда и обратно с минимальными энергетическими затратами [2, 3]. Также в последнее время все серьезней заявляет о себе проблема космического мусора, вследствие скопившихся на околоземных орбитах отработавших свой ресурс космических объектов и их частей. Поэтому появились идеи утилизации космического мусора путем использования тросов имеющих форму ленты. Суть заключается в том, что на вновь выводимые космические аппараты устанавливается механизм с тросом, который сначала находится в «дремлющем» состоянии до тех пор, пока космический аппарат выполняет свою миссию на орбите. По окончании миссии электронная аппаратура запускает процесс развёртывания тросовой системы. Тем самым это приводит к повышению величины аэродинамического торможения, в итоге осуществляется более быстрый спуск отработавшего свой срок объекта в нижние слои атмосферы, где тросовая система сгорает [4].

Идея экспериментов по утилизации отработавших свой срок объектов рассматривается также в работах [13, 14]. В статье [13] проводится анализ метода утилизации будущих космических объектов с использованием тросовых технологий. Здесь идея заключается в том, что к космическому объекту крепится устройство с тросом, который при работе объекта находится в свернутом состоянии. Однако когда рабочая миссия данного космического аппарата завершается, то производится развёртывание троса.

Здесь в отличие от идеи использования аэродинамического торможения (его увеличения за счет развёртывания ленты), предлагается использовать проводящий трос (проводник) с большим сопротивлением. В данной работе предложена схема создания сильного тока в тросе-проводнике, который в свою очередь будет индуцировать тепло, отчасти за счет энергии движения базового КА, которая будет переходить в энергию нагрева троса. Далее тепло будет рассеиваться в окружающую среду, тем самым будет происходить торможение КА, и его ускоренный переход в плотные слои атмосферы. Данная идея интересна тем, что позволяет на порядок уменьшить необходимую длину троса по сравнению с моделями, учитывающими и использующими только аэродинамическое торможение. Во второй статье [14], в отличие от первой, внимание акцентируется на утилизации уже существующего космического мусора. В работе проводится математическое описание модели захвата отработавших свой срок космических объектов путем захвата их в сети, прикрепленные тросом к космическому аппарату. Данный способ удобнее и безопаснее по сравнению с использованием жестких манипуляторов, так как сети могут иметь произвольно большой размер и представляют собой гибкую структуру.

2. Введение в динамику космических тросовых систем

Космические тросовые системы представляют собой механическую системы в общем случае с бесконечным числом степеней свободы, так как трос, соединяющий концевые тела представляет собой гибкую механическую структуру с конечной массой. В частных случаях, когда гибкостью и массой троса можно пренебречь, а также условие задачи позволяет рассматривать концевые тела как абсолютно твердые, можно существенно уменьшить число степеней свободы, рассматриваемой механической системы [15- 24].

В работе рассмотрим КТС состоящую из двух концевых тел имеющих форму материальной точки (первое концевое тело представляет собой космический аппарат (КА), второе спускаемую капсулу (СК)). Трос представляет собой весомую нить, которую мы будем моделировать системой материальных точек (рис. 3).

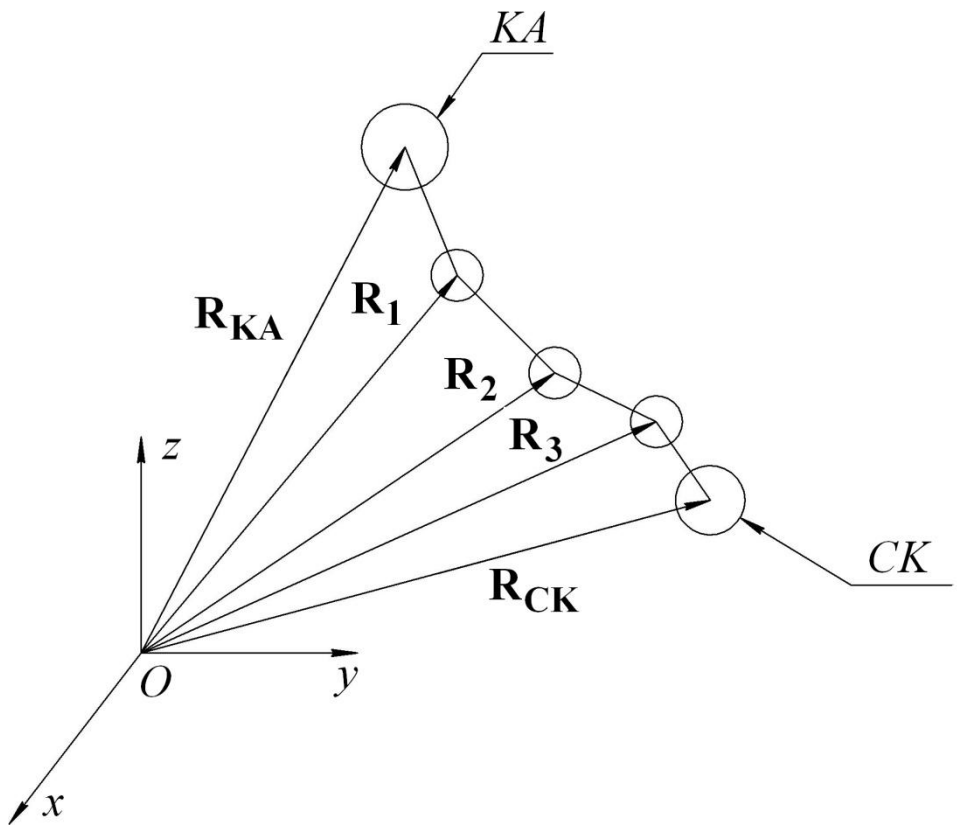


Рис. 3 – Схема гибкого троса

На рисунке 3 *Oxyz* - геоцентрическая инерциальная система координат, $\mathbf{R_{KA}}$ - радиус-вектор центра масс КА, $\mathbf{R_{СК}}$ - радиус-вектор центра масс СК,

$\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{R}_3$- радиус-вектора концов первого, второго, третьего участка соответственно. Чем больше рассматривается число участков, тем модель более приближена к модели гибкого троса (на рис. 3 приведёна схема троса, состоящего из четырёх участков).

Уравнения движения КТС, состоящей из n - участков, можно записать с использованием второго закона Ньютона, и имеют следующий вид:

$$\begin{aligned}
&\frac{d^2\mathbf{R_{KA}}}{dt^2} = -\frac{\mu\mathbf{R_{KA}}}{R_{KA}^3} - T_i\mathbf{e}_i,\\
&\frac{d^2\mathbf{R}_i}{dt^2} = -\frac{\mu\mathbf{R}_i}{R_i^3} + T_i\mathbf{e}_i - T_{i+1}\mathbf{e}_{i+1},\\
&\frac{d^2\mathbf{R_{CK}}}{dt^2} = -\frac{\mu\mathbf{R_{CK}}}{R_{CK}^3} + T_{i+1}\mathbf{e}_{i+1}.
\end{aligned} \qquad (1)$$

Здесь $R_{KA} = \left|\mathbf{R_{KA}}\right|$, $R_{CK} = \left|\mathbf{R_{CK}}\right|$, $R_i = \left|\mathbf{R_i}\right|$; $i = 1..n$, КА и СК можно присвоить индекс $i = 0$ и $i = n+1$; T_i - модуль силы упругости действующей на соответствующем участке троса; $\mathbf{e}_i$- единичный вектор определяющий ориентацию участка троса.

Сила упругости определяется выражением

$$T_i = \begin{cases} ES\dfrac{\left|\mathbf{R}_i - \mathbf{R}_{i+1}\right| - L_i}{L_i}, & if \quad \left|\mathbf{R}_i - \mathbf{R}_{i+1}\right| - L \geq 0 \\ 0, & if \quad \left|\mathbf{R}_i - \mathbf{R}_{i+1}\right| - L < 0 \end{cases},$$

где E - модуль упругости Юнга, S - площадь поперечного сечения троса, L_i - длина, соответствующая недеформированной длине участка троса.

Уравнения (1) описывают динамику развернувшейся тросовой системы, поэтому далее рассмотрим, динамику развертываемой тросовой системы. В процессе развёртывания длина троса изменяется, поэтому и изменяется число участков троса, из этого следует, что механическая модель описывается системой с переменным числом дифференциальных уравнений.

Уравнения по форме идентичны уравнениям системы (1), но их число в процессе развертывания меняется, поэтому систему (1) следует переписать в следующем виде:

$$
\begin{aligned}
&while(n \le n_{\max})\\
&\{\\
&while(L \le L_{nom})\\
&\{\\
&\frac{d^2\mathbf{R_{KA}}}{dt^2} = -\frac{\mu\mathbf{R_{KA}}}{R_{KA}^3} - T_i\mathbf{e}_i,\\
&\frac{d^2\mathbf{R}_i}{dt^2} = -\frac{\mu\mathbf{R}_i}{R_i^3} + T_i\mathbf{e}_i - T_{i+1}\mathbf{e}_{i+1},\\
&\frac{d^2\mathbf{R_{CK}}}{dt^2} = -\frac{\mu\mathbf{R_{CK}}}{R_{CK}^3} + T_{i+1}\mathbf{e}_{i+1},\\
&\}\\
&n = n+1;\\
&\}
\end{aligned}
\qquad (2)
$$

Здесь L - длина участка троса, прилегающая к КА; $L_{nom} = \frac{L_{\max}}{n_{\max}}$ - номинальная длина троса, где $L_{\max}$ - максимальная длина троса (длина троса в результате развертывания КТС) и $n_{\max}$ - максимальное число участков разбиения.

Для интегрирования системы (2) при каждом пересчёте числа участков троса n необходимо пересчитывать начальные условия интегрирования системы дифференциальных уравнений (2).

Исходные начальные условия используются для расчета КТС состоящей, только из одного участка, для этого используется следующая схема действий [25]:

1. Задается высота орбиты КА (H);
2. Задается начальная длина троса, как расстояние между точками крепления КА и СК (ΔL);

3. Задаются параметры орбиты: долгота восходящего узла Ω, наклонение орбиты i, угол поворота в плоскости орбиты $u = \omega + \nu$, который представляет собой сумму аргумента перицентра ω и истинной аномалии ν, эксцентриситет орбиты e, скорость отстрела СК V_{ots};
4. Определяется матрица перехода от геоцентрической системы координат к орбитальной системе координат, которая имеет вид:

$$A_\theta = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(i) & \sin(i) \\ 0 & -\sin(i) & \cos(i) \end{bmatrix},$$

$$A_u = \begin{bmatrix} \cos(u) & \sin(u) & 0 \\ -\sin(u) & \cos(u) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$\underset{geo \to orb}{A} = A_u A_i A_\theta =$$

$$= \begin{bmatrix} \cos(\theta)\cos(u) - \sin(\theta)\cos(i)\sin(u) & \sin(\theta)\cos(u) + \cos(\theta)\cos(i)\sin(u) & \sin(i)\sin(u) \\ -\cos(\theta)\sin(u) - \sin(\theta)\cos(i)\cos(u) & \cos(\theta)\cos(i)\cos(u) - \sin(\theta)\sin(u) & \sin(i)\cos(u) \\ \sin(\theta)\sin(i) & -cos(\theta)\sin(i) & \cos(i) \end{bmatrix}.$$

5. Определяем радиус вектор КА в геоцентрической системе координат:

$$\begin{bmatrix} R_{KA_x} \\ R_{KA_y} \\ R_{KA_z} \end{bmatrix} = \left[\underset{geo \to orb}{A} \right]^T \begin{bmatrix} R_z + H \\ 0 \\ 0 \end{bmatrix}.$$

6. Определяем радиус вектор СК в геоцентрической системе координат:

$$\begin{bmatrix} R_{CK_x} \\ R_{CK_y} \\ R_{CK_z} \end{bmatrix} = \begin{bmatrix} A \\ geo \to orb \end{bmatrix}^T \begin{bmatrix} R_z + H - \Delta L \\ 0 \\ 0 \end{bmatrix}.$$

7. Определяем фокальный параметр орбиты КА:

$$p = \left(R_z + H\right)\left(1 + e\cos(\nu)\right).$$

8. Определяем радиальную V_r и поперечную V_n компоненту скорости КА в проекциях на орбитальную систему координат:

$$V_r = \sqrt{\frac{\mu}{p}}\left(1 + e\cos(\nu)\right),$$

$$V_n = \sqrt{\frac{\mu}{p}} e\sin(\nu).$$

9. Определяем скорость КА в проекциях на оси геоцентрической системы координат:

$$\begin{bmatrix} V_{KA_x} \\ V_{KA_y} \\ V_{KA_z} \end{bmatrix} = \begin{bmatrix} A \\ geo \to orb \end{bmatrix}^T \begin{bmatrix} V_r \\ V_n \\ 0 \end{bmatrix}.$$

10. Определяем скорость СК в проекциях на оси геоцентрической системы координат:

$$\begin{bmatrix} V_{CK_x} \\ V_{CK_y} \\ V_{CK_z} \end{bmatrix} = \begin{bmatrix} A \\ geo \to orb \end{bmatrix}^T \begin{bmatrix} V_r - V_{ots} \\ V_n \\ 0 \end{bmatrix}.$$

Для интегрирования системы (2) при каждом пересчёте числа участков троса n необходимо пересчитывать начальные условия интегрирования системы дифференциальных уравнений (2).

После определения начальных условий для уравнений движения СК и КА в начальный момент времени развёртывания КТС. Необходимо

определить алгоритм добавления новой точки, который можно представить в следующем виде [18]:

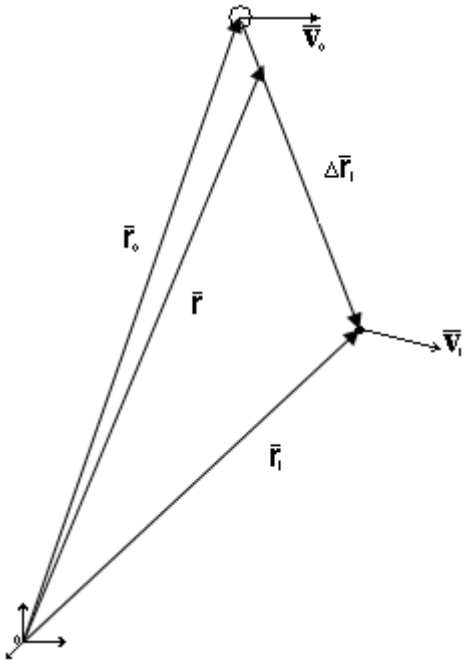


Рис. 4 – Схема добавления элемента КТС

На рис. 4 r_0 и v_0 радиус-вектор и скорость КА, r_1 и v_1 -радиус-вектор и скорость первой точки троса в геоцентрической системе координат. Положение этой точки относительно КА и ее относительная скорость находятся следующим образом:

$$\Delta\vec{r}_1 = \vec{r}_1 - \vec{r}_0\,,\ \Delta\vec{v}_1 = \vec{v}_1 - \vec{v}_0\,.$$

Положение новой точки относительно КА, определяется следующим образом:

$$\Delta\vec{r} = (|\Delta\vec{r}_1| - L)\frac{\Delta\vec{r}_1}{|\Delta r_1|} = (1 - \frac{L}{|\Delta r_1|})\Delta\vec{r}_1\,. \qquad (3)$$

Найдем скорость нового элемента. Обозначим через α угол между векторами $\Delta\vec{r}_1$ и $\Delta\vec{v}_1$. Затем найдем составляющую относительной скорости, направленную вдоль троса:

$$\vec{v}_{1p} = |\Delta\vec{v}_1|\cos\alpha\,\frac{\Delta\vec{r}_1}{|\Delta\vec{r}_1|}\,. \qquad (4)$$

Следовательно, составляющая нормальная к тросу может быть найдена как разность векторов:

$$\vec{v}_{1n} = \Delta\vec{v}_1 - \vec{v}_{1p}\,.$$

Тогда составляющая скорости новой точки, перпендикулярная тросу, определится из выражения:

$$\vec{v}_n = \frac{|\Delta\vec{r}|}{|\Delta\vec{r}_1|}\vec{v}_{1n}.$$

Откуда получим ее относительную скорость:

$$\Delta\vec{v} = \vec{v}_n + \vec{v}_{1p}. \qquad (5)$$

Учитывая соотношения (3) и (5), получим окончательные выражения для скорости и координат добавленного элемента:

$$\vec{v} = \vec{v}_0 + \Delta\vec{v}, \quad \vec{r} = \vec{r}_0 + \Delta\vec{r}. \qquad (6)$$

Добавление нового элемента приводит к добавлению шести дифференциальных уравнений, описывающих движение новой точки. Кроме того, для нового элемента значения выпущенной длины троса и скорости развертывания задаются следующим образом

$$\begin{aligned} V_{depl} &= \left|\vec{v}_{1p}\right| \\ L_{depl} &= L_{depl}^{prev} - L \end{aligned},$$

где L_{depl}^{prev} - выпущенная длина троса до добавления новой точки.

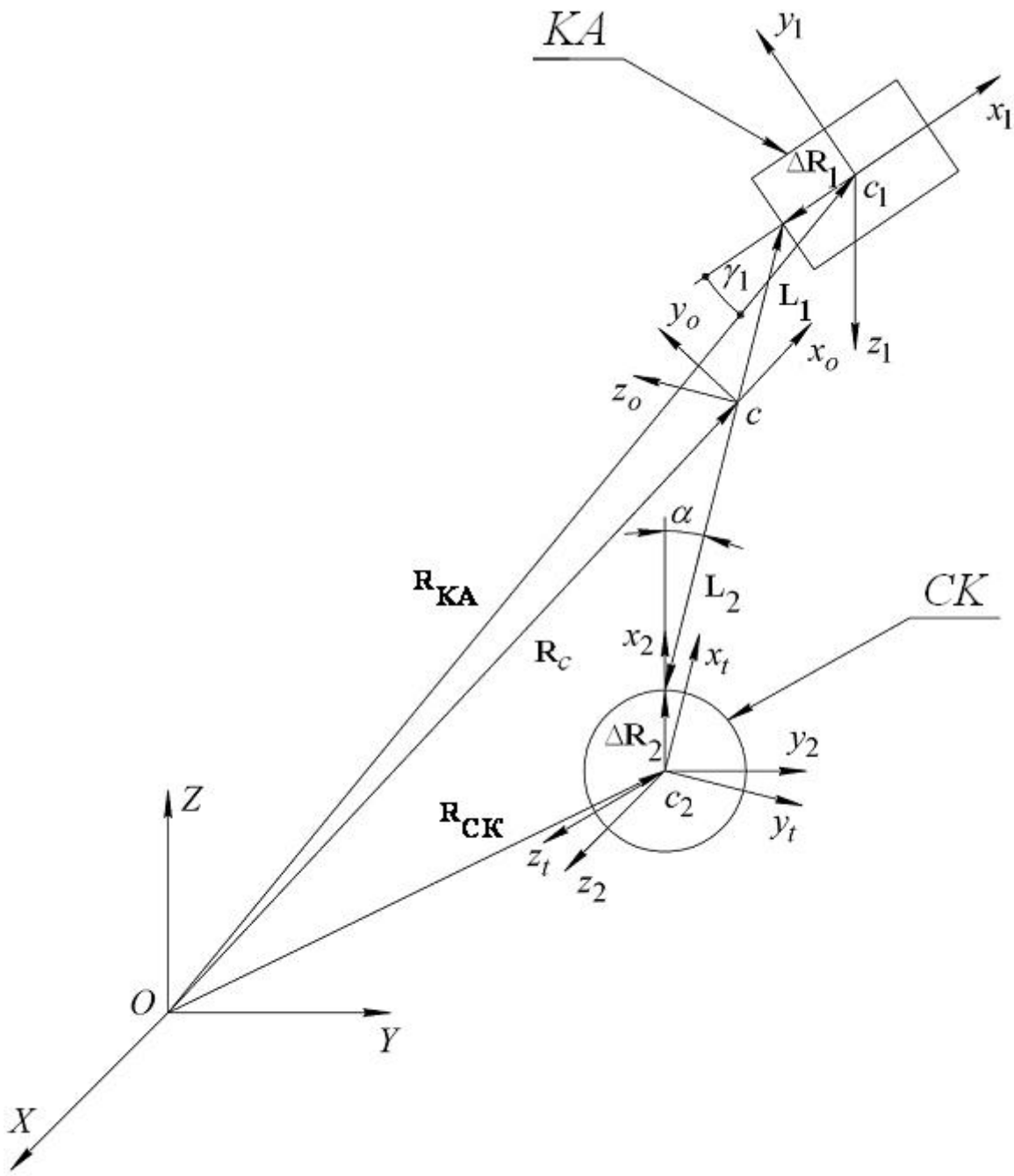
После добавления нового элемента взаимодействие между ним и КА соответствует выше описанному взаимодействию, а все элементы, которые расположены ниже него, рассматриваются так же, как для КТС постоянной длины (подробный анализ и вывод данной схемы, описан в [18]).

3. Динамика вращательного движения концевых тел

В случае развёртывания КТС, часто необходимо рассматривать концевые тела, как тела конечных размеров совершающих движения относительно центра масс. Данная проблема может возникнуть при проведении экспериментов в космосе, где необходимо выдерживать строгие ограничения на ориентацию, как базового КА, так и спускаемой капсулы (как например, в экспериментах серии OEDIPUS). Пример схемы иллюстрирующей сложность задачи описания движения КТС с учётом вращательного движения концевых тел приведён на рис. 5: На рис. 5 введены следующие обозначения: $\mathbf{R_{КА}}$ - радиус-вектор центра масс КА c_1; $\mathbf{R_{СК}}$ - радиус-вектор центра масс СК c_2; $\mathbf{R}_c$ - радиус-вектор центра масс связки c (КА, СК и троса); $\Delta\mathbf{R}_1$, $\Delta\mathbf{R}_2$ - радиус-вектора точек крепления троса к КА и СК соответственно; вектора $\mathbf{L}_1$, $\mathbf{L}_2$ определяют положение точек крепления троса к КА и СК относительно центра масс связки, соответственно, в работе полагается, что данные вектора коллинеарны; γ_1 угол между местной геовертикалью КА и его осью динамической симметрии x_1.

На рис. 5 изображены следующие системы координат:

1. $cx_oy_oz_0$ - подвижная орбитальная система координат, начало которой совпадает с центром масс связки КА, троса и СК. Ось cx_o - направлена по радиус-вектору центра масс связки, cz_o - направлена по вектору кинетического момента его орбитального движения, cy_o - направлена в сторону движения центра масс связки и дополняет систему координат $cx_oy_oz_0$ до правой.



*

Рис. 5 – Схема КТС

2. $OXYZ$ - неподвижная орбитальная система координат; O - центр притяжения (центр Земли). Неподвижная орбитальная система координат повернута относительно неподвижной геоцентрической СК на углы: Ω - поворот на восходящий узел, i - угол наклона орбиты (Приложение 1). При выводе уравнений не учитывается медленное изменение ориентация плоскости орбиты центра масс связки (величины $i,\ \Omega = const$).

3. $c_2x_ty_tz_t$ - тросовая система координат, ось c_2x_t параллельна вектору силы упругости троса $\mathbf{T}$, приложенной к СК; плоскость $c_2x_ty_t$ параллельна плоскости, проходящей через радиус-вектор точки крепления троса к КА и натянутый трос.

4. $c_1x_1y_1z_1$ - подвижная связанная с базовым КА система координат, оси которой направлены по главным осям инерции, центр совпадает с центром масс КА.

5. $c_2x_2y_2z_2$ - подвижная связанная со СК система координат, оси направлены по главным осям инерции, центр совпадает с центром масс СК.

Из рассмотрения рис. 6 можно заметить, что даже при допущении того, что трос представляет собой твердый стержень, задача описания данной механической системы весьма сложная. Так как необходимо учитывать движения центра масс системы по орбите, относительное движение троса, как стержня относительно орбиты центра масс, и наконец, движения СК и КА, как твёрдых тел относительно троса. При всём при этом надо понимать, что трос развёртывается и длина его переменная. Подробная схема вывода уравнения движения механической системы представленной на рис. 6 можно найти в [15].

В данном пособии рассмотрим пример описания вращательного движения концевых тел в более простой постановке, которая не учитывает подвижность тросовой системы координат.

Для описания динамики движения твердого тела относительно центра масс, будем использовать кинематические и динамические уравнения Эйлера. Динамические уравнения Эйлера имеют вид [26 - 30]:

$$\frac{d\tilde{\mathbf{K}}}{dt} + \boldsymbol{\omega} \times \mathbf{K} = \sum \mathbf{M}, \qquad (7)$$

где $\mathbf{K} = \mathbf{J}\boldsymbol{\omega}$ - кинетический момент во вращательном движении, где $\mathbf{J}$ - тензор инерции твердого тела; $\boldsymbol{\omega}$ - угловая скорость; $\sum \mathbf{M}$ - сумма моментов

внешних сил, знак тильды определяет локальную производную при дифференцировании в подвижной системе координат.

Проведём дифференцирование в (3) в итоге получим:

$$\mathbf{J}\frac{d\boldsymbol{\omega}}{dt}+\boldsymbol{\omega}\times\mathbf{J}\boldsymbol{\omega}=\sum\mathbf{M}. \qquad (8)$$

Разрешим уравнения (8) относительно производной $\frac{d\boldsymbol{\omega}}{dt}$ (угловых ускорений), в итоге получим:

$$\frac{d\boldsymbol{\omega}}{dt}=\mathbf{J}^{-1}\left(\sum\mathbf{M}-\boldsymbol{\omega}\times\mathbf{J}\boldsymbol{\omega}\right). \qquad (9)$$

Тензор инерции в выражении (9) представляет собой симметричную матрицу следующего вида:

$$\mathbf{J}=\begin{pmatrix} J_x & -J_{xy} & -J_{xz} \\ -Jxy & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{pmatrix},$$

где J_x, J_y, J_z - осевые моменты инерции; J_{xy}, J_{xz}, J_{yz} - центробежные моменты инерции.

В случае если тензор инерции твердого тела переменный (например, при изменении его геометрии или массы), то в уравнениях данное обстоятельство необходимо учесть дополнительным слагаемым $\frac{d\mathbf{J}}{dt}\boldsymbol{\omega}$, в итоге получим:

$$\frac{d\boldsymbol{\omega}}{dt}=\mathbf{J}^{-1}\left(\sum\mathbf{M}-\frac{d\mathbf{J}}{dt}\boldsymbol{\omega}-\boldsymbol{\omega}\times\mathbf{J}\boldsymbol{\omega}\right).$$

В случае исследования динамики космических тросовых систем выражения $\sum\mathbf{M}$ - представляет сумму момента от силы упругости троса и гравитационного момента, иногда при развёртывании КТС в нижних слоях атмосферы в данную сумму включается и аэродинамический момент. Хотя, как правило, аэродинамический и гравитационной момент много меньше на

два и более порядка чем момент от силы упругости троса, поэтому под суммой моментов внешних сил в дальнейшем будем понимать момент от силы упругости троса $\mathbf{M_{el}}$, который определяется следующим образом:

$$\mathbf{M_{el}} = \Delta\mathbf{r} \times \mathbf{T},$$

где $\Delta\mathbf{r}$ - радиус-вектор точки крепления троса относительно центра масс концевого тела, имеет вид $\Delta\mathbf{r} = \left(\Delta x,\ \Delta y,\ \Delta z\right)^T$, где $\Delta y,\ \Delta z$ - определяют параметры асимметрии точки крепления троса (см. рис. 6) $\Delta = \sqrt{\Delta y^2 + \Delta z^2}$, Δx - номинальное расстояние от центра масс СК до точки крепления троса; $\mathbf{T}$ - вектор силы упругости троса.

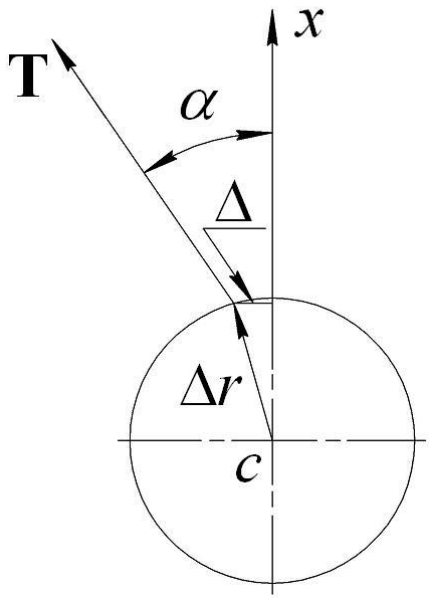


Рис. 6 – Схема крепления троса с асимметрией

К уравнениям (1.6) необходимо добавить кинематические уравнения Эйлера, которые имеют вид:

$$\frac{d\alpha}{dt} = \omega_z \cos\varphi + \omega_y \sin\varphi, \tag{10}$$

$$\frac{d\psi}{dt} = -\left(\omega_y \cos\varphi - \omega_z \sin\varphi\right)/\sin\alpha\ , \tag{11}$$

$$\frac{d\varphi}{dt} = \omega_x + \left(\omega_y \cos\varphi - \omega_z \sin\varphi\right) ctg\alpha\ . \tag{12}$$

Применяя систему уравнений (7), (10) – (12) для каждого концевого тела, а для расчёта силы упругости систему (1) либо (2), получим

динамическую модель КТС с учётом движения относительно центра масс концевых тел. Для исследования динамики вращательного движения концевых тел при развёртывании КТС можно применять и методы аналитической механики, например уравнения Лагранжа второго рода [15].

4. Динамика системы управления развёртыванием космической тросовой системы

Для выполнения поставленной задачи, при проведении экспериментов с КТС, как правило, необходимо чтобы трос развёртывался по определенному заранее заданному закону, например как в эксперименте YES2.

Управление развёртыванием троса можно реализовать за счёт управления его длиной и скоростью в каждый заданный момент времени. Управляющую силу можно сформировать в виде:

$$F_c = k_V \left(V - V_N\right) + k_L \left(L - L_N\right), \tag{13}$$

где k_V, k_L - коэффициенты обратной связи по скорости и длине троса, соответственно; V, L - текущая скорость и длина троса, соответственно; V_N, L_N - номинальная скорость и длина троса, соответственно. Номинальные параметры (V_N, L_N) могут быть получены из решения следующей системы уравнений [16]:

$$\frac{d\omega_\theta}{dt} = -2\frac{V_L}{L}(\omega_\theta + \Omega) - \frac{3}{2}\Omega^2 \sin 2\theta, \qquad \frac{d\theta}{dt} = \omega_\theta, \tag{14}$$

$$\frac{dV_L}{dt} = L\left[\left(\omega_\theta + \Omega\right)^2 - \Omega^2\left(1 - 3\cos^2\theta\right)\right] - \frac{T_0}{m_{CK}}, \qquad \frac{dL}{dt} = V_L, \tag{15}$$

где Ω - угловая скорость орбитального движения базового КА по круговой орбите, m_{CK} - масса СК; θ - угол отклонения троса от направления местной геовертикали КА; L, V_L - длина и скорость развёртывания троса, T_0 - номинальная сила упругости троса; ω_θ - угловая скорость вращения троса относительно базового КА.

Номинальную силу упругости можно задать динамическим законом, к примеру, для расчетов будем использовать динамический закон близкий к

тому, который использовался в эксперименте YES2(в данном эксперименте процесс развёртывания состоял из двух временных этапов, первый длился 6000 секунд, второй 2165 секунд):

$$T_0 = m \cdot \Omega^2 \cdot \left(a \cdot L + b \cdot \frac{V_L}{\Omega} - c \cdot L_k \right), \qquad (16)$$

где $a = 4,6$, $b = 3,5$, $c = 1,6$ - параметры закона управления; $L_k = 3000$ м - конечная длина троса для первого этапа. Тогда программные зависимости для длины $L_p(t)$ и скорости $V_{Lp}(t)$ определяются интегрированием системы (10) – (11) при заданных начальных условиях, например, $\omega_\theta(0) = \theta(0) = 0$, $V_L(0) = 2.5$ м/с, $L(0) = 0,1$ м.

Второй этап развёртывания КТС состоит из участков быстрого разгона и торможения, и длится около 2165 секунд. На этом этапе используется релейный закон управления [41]:

$$\begin{array}{l} T_0 = T_{\min}, \quad \text{если} \quad t \le t_n \\ T_0 = T_{\max}, \quad \text{если} \quad t > t_n \end{array}, \qquad (17)$$

где $T_{\min}$, $T_{\max}$, t_n - параметры закона; t_n - время переключения. Выбором значений параметров закона можно обеспечить заданные конечные значения характеристик движения капсулы. В данной работе использовались следующие значения: $T_{\min} = 0,02$ Н, $T_{\max} = 2,1$ Н, $t_n = 7900$ с, что соответствует $\omega_{\theta k} = V_{Lk} = 0$, $\theta_k = -56^\circ$, $L_k = 29500$ м.

Соединение законов (16) и (17) дает функцию $T_0(t)$, имеющую две точки разрыва (точки релейного переключения). С другой стороны реализация релейной зависимости с помощью системы регулирования вряд ли возможна из-за инертности любой механической системы. Поэтому сгладим зависимость $T_0(t)$, обеспечив плавные переходы, например, с помощью функции $T_0(t) = T_{-0} + \left(T_{+0} - T_{-0}\right) \cdot \sin^2\left[k_n \left(t - t_{n1}\right)\right]$, где T_{-0} и T_{+0} - значения силы слева и справа от точки разрыва, $t \in \left[t_{n1}, t_{n2}\right]$,

$t_{n1,2} = t_n \mp \frac{\pi}{4k_n}$, k_n - коэффициент, определяющий гладкость сопряжения участков.

Реальную длину и скорость троса можно определить из решения уравнений динамики механизма управления, которые имеют вид:

$$\begin{aligned} \frac{dL}{dt} &= V, \\ \frac{dV}{dt} &= \frac{1}{J_k}\left(T - F_c\right), \end{aligned} \tag{18}$$

где J_k - инерционность механизма управления.

В данной разделе был проведено краткое описание динамики космических тросовых систем, в следующем разделе рассмотрим как с использованием пакета PGraph, можно провести моделирования развёртывания КТС с распределёнными параметрами.

5.Моделирование развёртывания космической тросовой системы в пакете PGraph

PGraph представляет среду графического программирования. В основе программирования лежит идея создания модулей выполняющих определённые действия. Программирования в среде PGraph требует построения графа решений исходной задачи, причем вершины графа определяются акторами, а ветви графа предикатами.

Актор представляет собой текст кода программы написанный на языке C/C++ [31 - 38], предикат определяет условия перехода от одной вершины графа к другой.

Для программирования в среде PGraph, как правило, необходимо задать словарь данных, который определяет используемые типы данных и глобальные переменные, с использованием которых и производится обмен данными между вершинами. Каждая вершина графа определяет собой отдельную функцию, поэтому все данные объявленные внутри актора, имеют локальную область видимости.

В среде PGraph можно подключать заголовочные файлы сторонних библиотек C\C++, поэтому в своих программах можно использовать например возможности Standard Template Library (STL – стандартная библиотека шаблонов) [36, 37], а также операционно-зависимые библиотеки например библиотеку функций Win32[39 - 41]. В заголовочных файлах можно определять собственные пользовательские типы (классы), а также можно определять глобально перегруженные операторы (например операцию умножения матриц, векторов и т.д.)

Для расчёта необходимо указать корневую вершину графа, которая определяет точку входа в программу. Поэтому для моделирования в среде PGraph, необходимо, сначала определить типы и глобальные данные, затем определить и описать структуру вершин графа (акторы), в конечном итоге

определить условия перехода от одной вершины к другой (предикаты) и корневую вершину.

Общая схема программы моделирования динамики КТС, представлена на рис. 7:

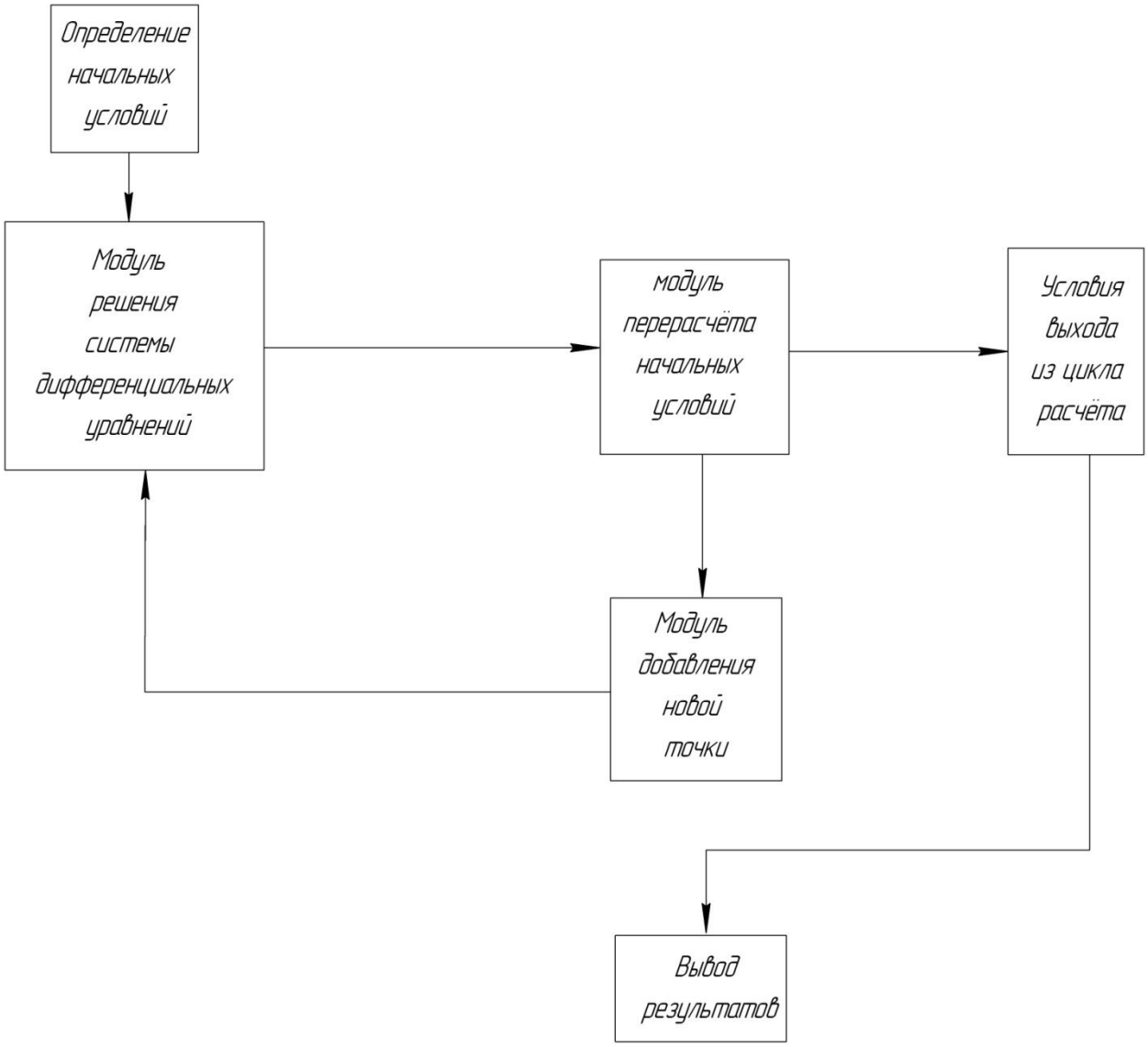


Рис.7 – Схема модулей расчёта динамики КТС

На рис. 7 изображены основные модули необходимые для моделирования процесса развёртывания КТС.

1. Модуль определения начальных условий, определяет начальные условия для КТС состоящей из двух концевых тел, и троса, который представлен, в данный момент времени, одним участком. Данный модуль может содержать расчёт с использованием формул, раздела 2, либо задания начальных условий по умолчанию.

2. Модуль решения системы дифференциальных уравнений представляет собой реализацию численного алгоритма решения задачи Коши для системы дифференциальных уравнений. Данный модуль можно представить в виде следующей вложенной схемы модулей, представленной на рис. 8, которая содержит модуль описания правых частей ОДУ, где необходимо описать уравнения движения КТС, а также модуль описания метода интегрирования задачи Коши.

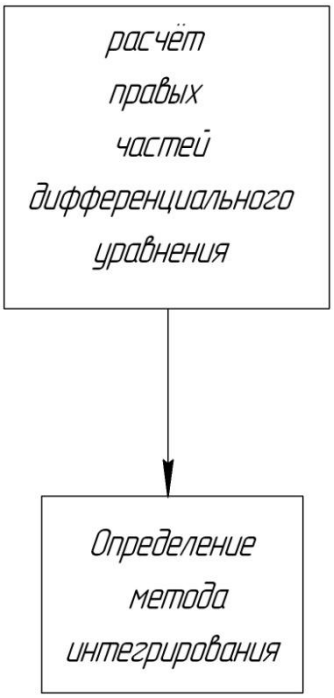


Рис. 8 – Схема модуля интегрирования систем ОДУ

3. Модуль перерасчёта начальных условий, содержит описания алгоритма расчёта начальных условий для нового участка (новой добавляемой точки) и возможно изменения начальных условий для исходной системы ОДУ.
4. Модуль добавления новой точки, реализует алгоритм изменения длины вектора правых частей уравнений (увеличения числа уравнений в системе, для рассматриваемой задачи, добавления нового участка увеличивает число уравнений на 6).

5. Условия выхода из цикла расчёта определяют предикат остановки итерационного процесса интегрирования и последовательного добавления новых участков троса.
6. Модуль вывод результатов, необходим для получения сведений о проведённом расчёте.

Общий порядок проектирования в среде граф изображён на рис. 1.9:

Рис. 9 – Схема проектирования модели развёртывания КТС в среде PGraph

5.1 Словарь данных для моделирования динамики космической тросовой системы

Для моделирования динамики КТС, необходимо задать основные типы которыми будет удобно пользоваться при описании акторов и предикатов. Так как уравнения механики описываются векторными величинами, поэтому за основу удобно взять один из последовательных контейнеров. Последовательные контейнеры в C++ описываются следующими классическими структурами данных: массивы, вектора, двусвязанные списки, двусторонние очереди, а также их адаптерами – стеки, очереди с приоритетами.

Из рассмотрения уравнений движения (2) следует, что при моделировании троса как модели с распределёнными параметрами, система

уравнений будет изменять свою размерность, поэтому логично для описания данных использовать последовательный контейнер, который позволяет максимально удобно изменять свой размер.

Проведём анализ последовательных контейнеров по данному критерию:

1. Массив представляет собой структуру данных, размер которой определяется в момент его создания. Поэтому использовать непосредственно массив будет неудобно.
2. Вектор представляет собой структуру, которая как раз позволяет работать с динамически изменяемыми данными, то есть размер вектора легко изменить, добавив в него новый элемент, причем вектор библиотеки STL уже имеет по умолчанию запрограммированные шаблонные методы добавления нового элемента (функции push_back – добавление в конец, insert добавления в произвольное место вектора). Сложность операции добавления нового элемента O(n) при операции insert.
3. Двусвязанные списки представляют структуру, которая легко позволяет добавлять новый элемент в любое место, причем сложность данной операции константная! Тем самым с точки зрения добавления нового элемента, двусвязный список максимально эффективен.
4. Двусторонняя очередь, также как и вектор позволяет добавлять элементы, имеет одно, но несущественное преимущество имеет по умолчанию функции добавления и удаления элемента из начала очереди (за константное время).
5. Использование адаптеров последовательных контейнеров является в данном вопросе неактуальной, так как они не дают каких-либо новых преимуществ.

Из проведенного выше анализа можно выделить, что с точки зрения вставки элемента удобнее всего использовать двусторонний список. Двусторонняя очередь и вектор из библиотеки STL в данном случае эквивалентны.

Так как двусторонняя очередь и вектор с точки зрения операции вставки эквивалентны, далее будем рассматривать только вектор, так как он более употребительный последовательный контейнер, поэтому в дальнейшем будем сравнивать вектор и двусвязный список.

При записи уравнений движения нам потребуется, очень часто обращаться к произвольным элементам последовательного контейнера. Из этого следует, что нам нужна структура данных имеющая произвольный доступ к элементам контейнера. Исходя из этого требования, двусвязный список нам не подходит, так как он не поддерживает итератор произвольного доступа, и не позволяет обращаться к своим элементам (двусвязный список STL поддерживает двунаправленный итератор, который позволят последовательно обходить весь список, тем самым получаем сложность доступа O(n) [42]). Вектор позволяет за константное время обращаться к любому своему элементу.

Из проведенного анализа следует, что вектор обладает в совокупности максимальными преимуществами и его удобно взять как основную структуру данных при описании систем уравнений динамики КТС.

С использованием вектора также удобно строить массивы произвольной размерности. Пример задания и вывод матрицы с использованием шаблона vector приведён в следующем коде, написанном с использованием Microsoft Visual Studio 2012:

```
#include <vector>
#include <iostream>

int main()
{
 std::vector<double> V1(10,5), V2(10, 7);    // Определяем два вектора по 10 элементов
 std::vector<std::vector<double> > VDouble;  // Определяем матрицу
 /* Заполняем матрицу заданными векторами */
 VDouble.push_back(V1);
 VDouble.push_back(V2);
```

```
 /* Выводим полученную матрицу */
 for(int i = 0; i<VDouble.size(); i++)
    {
     for(int j = 0; j<VDouble[i].size(); j++)
           {
           std::cout<<VDouble[i][j]<<" ";
           }
     std::cout<<std::endl;
    }
return 0;
}
```

Все компоненты библиотеки STL находятся в пространстве имен std, поэтому необходимо либо использовать директиву using namespace std, либо непосредственно подключать данное пространство имён к каждому объекту, как приведёно в коде. Хотелось бы остановится на записи вида `std::vector<std::vector<double> >` перед второй закрывающей скобкой, как правило, нужен пробел, хотя если использовать максимально современный компилятор, который поддерживает стандарт C++ 2011, то коррекция угловых скобок не нужна, соответственно запись `std::vector<std::vector<double>>` не будет содержать синтаксической ошибки.

Теперь необходимо проанализировать, какие переменные и типы данных нам нужны.

Расчёты с числовыми данными при интегрировании сложных систем дифференциальных уравнений должны обеспечивать широкий диапазон изменений и высокую точность, соответственно данным условиям подходит встроенный тип с плавающей точкой double.

Из проведённого анализа следует, что для решения поставленной задачи моделирования динамики КТС, удобно использовать шаблонный класс vector<> специализированный встроенным типом double, так как данная схема обеспечивает высокую гибкость за счёт возможности использовать встроенные алгоритмы STL, высокую скорость работы и точность вычислений.

5.2 Перегрузка операций и её использование для решения систем ОДУ

При расчёте в динамике КТС, часто приходится иметь дело с многомерными массивами данных, и как было показано выше, для программирования расчётных задач, удобно использовать структуру данных vector<> и её производные (например, vector<vector<> >). К примеру, над переменными пользовательского типа приходится выполнять операции сложения, умножение на целые числа, вычитание. По умолчанию данные операции предусмотрены для встроенных типов данных. Из этого следует, что операции для собственных типов данных, необходимо определить пользователю в процессе программирования (перегрузить данные операции). Перегрузка операций в C++ реализуется с использованием операторной функции, символически которую можно записать следующим образом:

$$Tout\ operatorX(Tin,...) \qquad (1.19)$$

Здесь $Tout$ - тип переменной возвращаемой в результате проведённых действий; Tin - тип переменной принимаемой функцией; X - определение операции, например $\{+,-,*,/,...\}$; многоточие, определяет, что список может содержать более одной переменной (возможно две переменные – для бинарных операций (но не более двух)).

Число переменных в операторной функции (1.19) зависит как от типа операции, так и от того глобальная ли это функции или это функция является членом класса [31 - 38].

Использовать (1.19), как функцию класса, при программировании в среде PGraph неудобно, так как чаще всего все основные переменные являются глобальными, поэтому при перегрузке необходимо использовать её глобальную форму, пример использования для суммы двух матриц приведён в листинге:

```
inline vector<vector<double> > operator+(vector<vector<double> > const &VDouble1,
vector<vector<double> > const &VDouble2 )
{

	size_t vsizestrok = VDouble1.size();
	size_t vsizestolb = VDouble1[0].size();
	vector<vector<double> > Vsum(VDouble1);
	auto_ptr<vector<vector<double> > > p1(new vector<vector<double> >);
	//Обработка возможных исключений
	try
```

```
        {
        if(VDouble1.size() != VDouble2.size() || VDouble1[0].size() != VDouble2[0].size())
              throw p1;
        else //Проводим расчёт суммы двумерных массивов
            {
            for(int i = 0; i<vsizestrok; i++)
                  {
                  for(int j = 0; j<vsizestolb; j++)
                     {
                     Vsum[i][j] = VDouble1[i][j] + VDouble2[i][j];
                     }
                  }//конец внешнего цикла for
             }
        }// Конец try
        catch(auto_ptr<vector<vector<double> > >)
        {
            cout<<"Error operator+ (type: vector<vector<double> >)"<<endl;
        }

return Vsum;
}
```

Проблема использования перегруженных операций в среде PGraph, заключается в том, что каждый модуль имеет свою локальную область видимости, а данные операции необходимо определить глобально! Это можно сделать, если определить заголовочный файл, подключаемый к программе, реализованной на PGraph, где требуется разместить определение перегруженной операции. К тому же именно данный способ позволяет подключать и заголовочные файлы библиотеки STL. Хотя данный способ на первый взгляд прост он таит, одну серьезную проблему, при генерации кода для каждого актора среда PGraph генерирует свой файл исходного кода, куда добавляет всё подключенные заголовочные файлы. Так как в заголовочном файле мы размещаем определение, а не только объявление операторной функции, то тем самым мы нарушаем правило одного определения, что приведёт к ошибке компоновки. Для того чтобы избежать данной ошибки можно операторную функцию определить как встраиваемую, что и сделано в листинге (функция определена с идентификатором inline). Другой способ решения данной проблемы использовать безыменное пространство имён [32, 38].

5.3 Реализация модуля расчёта начальных условий

Расчёт или задания корректных начальных условий необходим, в задачах связанных с расчётом динамических систем является важной задачей вследствие того, что некорректные начальные условия могут привести к крайне недостоверным результатам (к результатам, противоречащим физики процесса). Рекомендуется непосредственно при проектировании решения динамической системы в среде PGraph непосредственно реализовывать расчёт начальных условий в отдельном модуле. Пример схемы реализации для рассматриваемой системы, моделирующей развёртывания КТС, приведён на рис. 10:

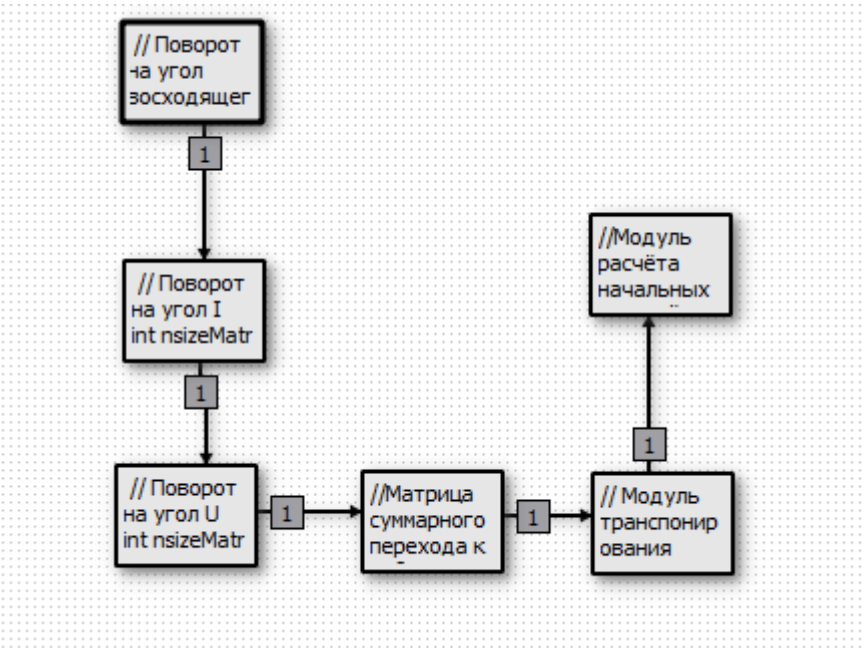


Рис. 10 – Схема расчёта начальных условий динамики развёртывания КТС в PGraph

На схеме изображена последовательность акторов (сплошной чёрной линией обведена корневая вершина), первые три вершины определяют матрицы поворота, необходимые для определения суммарной матрицы перехода от геоцентрической к орбитальной, которая в свою очередь определяется в четвёртом акторе. Так как параметры орбиты удобно

определять в орбитальной системе координат, а уравнения движения КТС мы записываем в неподвижной системе координат. Тем самым необходимо определить матрицу обратного перехода от орбитальной к геоцентрической системе координат (реализуется в пятом акторе). В шестом акторе производится расчёт начальных условий по формулам описанным в разделе 2 данного пособия. Замечание: здесь нумерация акторов идёт от корневой вершины.

Все предикаты в рассматриваемой схеме принимают значения true, то есть всегда реализуется последовательный переход от одной вершины графа к другой.

5.4 Реализация модуля интегрирования

Моделирование процесса развёртывания КТС требует решения систем обыкновенных дифференциальных уравнений (ОДУ). Существует достаточно большое количество численных методов решения ОДУ [43-45] (явные, неявные методы Рунге-Кутты, методы прогноз коррекции и т.д.), но наиболее распространенные и удобные для практической реализации являются явные методы Рунге-Кутты. Удобство явных методов Рунге-Кутты заключается в том, что не требуется вычислять производные высоких порядков, и решать системы алгебраических уравнений, как в неявных методах.

Самым распространенным методом является метод Рунге-Кутты четвёртого порядка, он прост в использовании и обеспечивает высокую точность интегрирования - глобальная ошибка $O(h^4)$, имеет следующую реализацию [45]:

$$
\begin{aligned}
&y_{k+1} = y_k + w_1 k_1 + w_2 k_2 + w_3 k_3 + w_4 k_4,\\
&k_1 = hf(t_k, y_k),\\
&k_2 = hf(t_k + a_1 h, y_k + b_1 k_1),\\
&k_3 = hf(t_k + a_2 h, y_k + b_2 k_1 + b_3 k_2),\\
&k_4 = hf(t_k + a_3 h, y_k + b_4 k_1 + b_5 k_2 + b_6 k_3).
\end{aligned}
\qquad (20)
$$

Здесь h - шаг интегрирования; f - функция правых частей ОДУ; t - аргумент неизвестной функции y; остальные неизвестные величины определяют коэффициенты метода Рунге-Кутты.

Коэффициенты метода Рунге-Кутты должны соответствовать решению следующей системы уравнений:

$$
\begin{aligned}
&b_1 = a_1,\\
&b_2 + b_3 = a_2,\\
&b_4 + b_5 + b_6 = a_3,\\
&w_1 + w_2 + w_3 + w_4 = 1,\\
&w_2 a_1 + w_3 a_2 + w_4 a_3 = \frac{1}{2},\\
&w_2 a_1^2 + w_3 a_2^2 + w_4 a_3^2 = \frac{1}{3},\\
&w_2 a_1^3 + w_3 a_2^3 + w_4 a_3^3 = \frac{1}{4},\\
&w_3 a_1 b_3 + w_4 (a_1 b_5 + a_2 b_6) = \frac{1}{6},\\
&w_3 a_1 a_2 b_3 + w_4 a_3 (a_1 b_5 + a_2 b_6) = \frac{1}{8},\\
&w_3 a_1^2 b_3 + w_4 \left(a_1^2 b_5 + a_2^6 b_6\right) = \frac{1}{12},\\
&w_4 a_1 b_3 b_6 = \frac{1}{24}.
\end{aligned}
\qquad (21)
$$

Система содержит 11 уравнений с 13-ю неизвестными. Следует добавить два дополнительных условия, чтобы решить систему. Чаще всего используют значения $a_1 = \frac{1}{2},\ b_2 = 0.$ В итоге получим:

$$a_2 = \frac{1}{2},\ a_3 = 1,\ b_1 = \frac{1}{2},\ b_3 = \frac{1}{2},\ b_4 = 0,$$
$$b_5 = 0,\ b_6 = 1,\ w_1 = \frac{1}{6},\ w_2 = \frac{1}{3},\ w_3 = \frac{1}{3},\ w_4 = \frac{1}{6}. \quad (22)$$

С учётом (22) перепишем (20) следующим образом:

$$\begin{aligned} k_1 &= f(t_k, y_k), \\ k_2 &= f(t_k + \frac{h}{2}, y_k + \frac{h}{2}k_1), \\ k_3 &= f(t_k + \frac{h}{2}, y_k + \frac{h}{2}k_2), \\ k_4 &= f(t_k + h, y_k + hk_3), \\ y_{k+1} &= y_k + \frac{h(k_1 + 2k_2 + 2k_3 + k_4)}{6}. \end{aligned} \quad (23)$$

Используя метод (23) можно проинтегрировать систему дифференциальных уравнений практически любой сложности. Опишем реализацию данного метода в среде PGraph.

Для реализации метода Рунге-Кутты четвёртого порядка необходимо определять четыре коэффициента $(k_1,\ k_2,\ k_3,\ k_4)$ на каждом шаге интегрирования и при помощи их определять новое значения искомой функции $y(t)$. Так как исходя из формул (15) данные коэффициенты могут быть рассчитаны различным способом, то вычисления каждого коэффициента $(k_i,\ i = 1..4)$ необходимо проводить в отдельном модуле.

При вычислении коэффициентов $(k_i,\ i = 1..4)$ и искомой функции требуется знать вид правых частей, но если реализацию метод Рунге-Кутты жёстко привязать к правым частям какого-либо дифференциального уравнения, то это приведёт к потере гибкости и сложностям в дальнейшем сопровождении и использовании. Из этого следует, что метод Рунге-Кутты необходимо реализовать отдельно от правых частей ОДУ с использованием какого-либо блока связки.

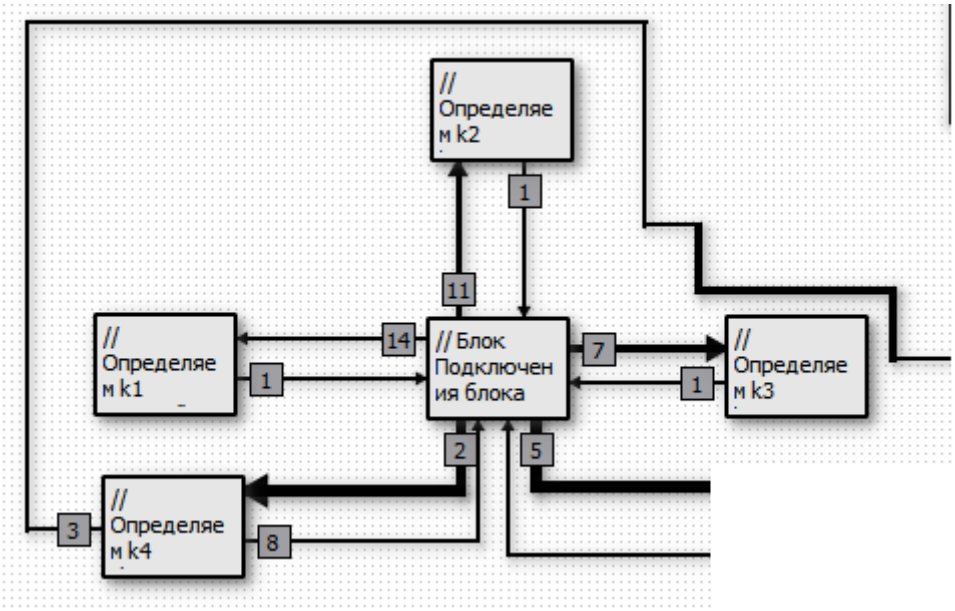


Рис. 11 – метод Рунге-Кутты четвёртого порядка точности

Из рис. 11 видно, что реализация метода Рунге-Кутты содержит пять акторов и 11 предикатов рассмотрим их подробней.

Реализация акторов $\left(k_i,\ i=1..4\right)$:

{ // Определение k1
k1 = A;
*gY = gY + (step/2.0)*k1;*
gt = gt + (step/2.0);
gKbool = 1;}

{ //Определение k2
k2 = A;
*gY = gY - (step/2.0)*k1;*
*gY = gY +(step/2.0)*k2;*
gKbool = 1;}

{// Определение k3
k3 = A;
gY = gY - (step/2.0)*k2;
gY = gY +(step)*k3;
gt = gt + (step/2.0);
gKbool = 1;}

{// Определение k4
k4 = A;
gY = gY - (step)*k3;
gY = gY + (step/6.0)*(k1+k2+k3+k4); *gKbool = 1;* }

Здесь k1, k2, k3, k4, A, gY глобальные переменные соответствующие значениям$\left(k_i,\ i=1..4,\ f(\ ),\ y\right)$ в (23). Здесь A значения вектора правых

частей ОДУ. Глобальная переменная gKbool типа int (целочисленный тип) необходима для определения предикатов перехода от модуля правых частей к модулю присоединения блока интегрирования.

В блоке подключения может располагаться любой программный код, например обработка возможных ошибок в процессе интегрирования.

Предикаты, в свою очередь, должны определять момент входа в тот или иной блок расчёта коэффициентов k1, k2, k3, k4. Расчёт коэффициентов должен быть проведён в следующем порядке: сначала k1затем k2 далее k3 и наконец k4. Для определения времени входа в модули расчёта коэффициентов, удобно использовать глобальную переменную, принимающую значения от 1 до 4, что соответствует расчёту нужного в данный момент коэффициента $(k_i,\ i=1..4)$. Данная переменная должна быть видна всем модулям, поэтому её следует задать на глобальном уровне. Её значение удобно изменять в модуле вычисления правых частей ОДУ.

Проведённое описание предикатов иллюстрирует следующий листинг:

```
{gk==1; // условие перехода в модуль k1}
{gk==2; // условие перехода в модуль k2}
{gk==3; // условие перехода в модуль k3}
{gk==4; // условие перехода в модуль k4}
```

Предикаты, описанные в листинге, определяют условия перехода из “блока подключения блока интегрирования” в модули проводящие расчёт коэффициентов $(k_i,\ i=1..4)$. Предикаты, которые определяют обратный переход из модулей $(k_i,\ i=1..4)$ в “блок подключения блока интегрирования”, определены - 1, т.е. при любых данных имеют значение true.

Модуль расчёта k_4 - содержит ещё один предикат (обозначен цифрой 3 на рис. 11), который должен определять дополнительные условия (например, реализацию обратной связи).

Метод Рунге-Кутты четвертого порядка в среде PGraph можно реализовать иначе. Как было описано выше, модуль расчёта вектора правых частей допустимо внедрить непосредственно в решение задачи, например, таким образом, как показано на рис. 12.

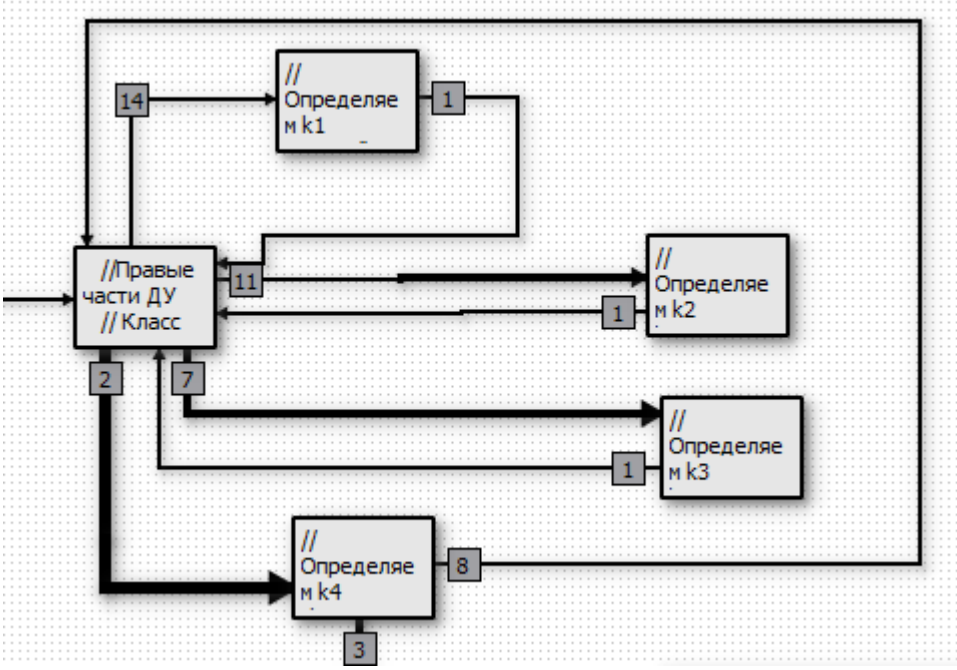


Рис. 12 – Метод Рунге-Кутты четвертого порядка реализация в среде PGraph

5.5 Реализация вектора правых частей обыкновенных дифференциальных уравнений

При определении правых частей дифференциальных уравнений нам может понадобиться задавать сложные расчётные формулы, поэтому для их записи часто требуется описать какие либо отдельные функции реализации. К примеру, в динамике КТС требуется определять силу упругости троса, управляющую силу, либо при расчёте движения относительно центра масс моменты от силы упругости троса. Тем самым нам удобно не записывать длинные выражения, а использовать отдельную функцию для расчёта той или иной величины. Проблема заключается в том, что непосредственно

описывать данную функцию в другом модуле трудоемко и громоздко, для каждой тривиальной операции придётся создавать свой актор и предикаты перехода и выхода из него. Тем самым возникает желание как-то определить функцию внутри актора, но как известно актор сам уже представляет собой функцию, а программа разработанная в среде PGraph транслируется в C++ код. В то же время, известно, что в C++ недопустимо использовать вложенные функции. Решить данную проблему можно, если учесть что в C++ можно определять классы локально, то есть внутри тела функции. В классах в свою очередь можно определять функции, в том числе и статически. Обращение к статическая функции в C++ не требует создания объекта экземпляра класса, тем самым можно реализовывать функции локально используя для этого пространство имён класса.

Пример листинга использования статических функций:

```
#include <iostream>
using namespace std;

int main()
{
  class StaticFunctUse
  {
  public:
        static double Summa(double const& a, double const& b)
        {
               double c = a+b;
               return c;
        }

        static double Proizv(double const& a, double const& b)
        {
               double c = a*b;
               return c;
        }
  };

 double a = 10;
 double b = 12;
 double c = StaticFunctUse::Proizv(a,b);
 double d = StaticFunctUse::Summa(a,b);

 return 0;
}
```

При использовании статических функций класса их необходимо определять в местах их объявления, то есть внутри класса.

На статические функции члены класса в C++ накладывают следующие ограничения:

1. Статическая функция не может быть константной;
2. Статическая функция не может быть виртуальной;
3. Статическая функция, определённая в локальном классе не может быть шаблонной (замечание, любая функция класса, если он объявлен в локальной области видимости не может быть шаблонной функцией).

В то же время для локальных классов и их статических функций в C++ определены операции наследования, т.е. идентификатор static наследуется как для глобально, так и для локально определённых функций.

Теперь перейдём к вопросам реализации вектора правых частей при исследовании динамики КТС. Будем рассматривать реализацию уравнений (2). Как было описано выше, данная система уравнений имеёт переменное число уравнений, поэтому для её записи удобно использовать последовательный контейнер (вектор). Так как при моделировании развёртывания троса, непрерывная механическая среда (трос) заменяется на дискретную механическую систему, состоящую из совокупности материальных точек.

Из теоретической механики известно, что каждая материальная точка имеет три степени свободы, поэтому для описания её движения в пространстве требуется три ОДУ второго порядка, что соответствует шести уравнениям первого порядка (метод Рунге-Кутты позволяет интегрировать только дифференциальные уравнения первого порядка).

В начальный момент развёртывания (при формировании первого участка), требуется 18 дифференциальных уравнений (в случае если мы не учитываем движение относительно центра масс концевых тел, иначе требуется 30 уравнений). Шесть уравнений требуется для описания движения центра масс КА, шесть уравнений для описания движения центра масс СК и шесть уравнений для расчёта номинальной программы развёртывания

совместно с моделированием работы механизма управления. В момент появления каждого нового участка система увеличивает число своих уравнений на 6 и требует перерасчета начальных условий.

Исходя из изложенного выше, рассмотрим, сначала, как реализовать вектор правых частей при наличии только одного участка, код актора представлен в листинге:

```
//Правые части ДУ
gk++;
if(gk>4)
gk= 1;
// Класс для расчёта силы упругости и управления
class Forces
{
public:
// Модуль вектора
static double ModVect(double x, double y, double z)
{
double r = sqrt(x*x + y*y + z*z);
return r;
}
// Расчёт силы упругости
static double Fypr(double Ll, double fkat)
{
double Fy;
double rkatl = 1.0;
double cl = 1000;
if ( (Ll-fkat*rkatl)>0)
{Fy = cl*(Ll-fkat*rkatl)/(fkat*rkatl);}
else
{Fy = 0;}
return Fy;
}
// Расчёт номинальной силы упругости
static double Fc(double V, double Ll, double t)
{
double a = 4.6, b = 3.5, cc = 1.6, Lk = 3000;
double mckl = 20;
double PSIl = 0.00115872474917774;
double PIl = 3.141592653589793;
double Fcopr = mckl*PSIl*PSIl*(a*Ll+b*(V/PSIl)-cc*Lk);
double kn = 0.002, tp = 1700, Tmin = 0.002, Tmax = 1.75, T1 = 6000, kp = 0.002, Tkl = 0.243;
double t1 = tp - PIl/(4*kp);
double t2 = tp + PIl/(4*kn);
double t2p = T1 + PIl/(4*kp);
double t1p = T1 - PIl/(4*kp);
double Fc1;
```

```
if(t<t1p)
Fc1 = Fcopr;
if((t>t1p) && (t<t2p))
Fc1 = (Tmin - Tkl)*pow(sin(kp*(t-t1p)),2) + Tkl;
if((t>t2p) && (t<t1+T1))
Fc1 = Tmin;
if((t>t1+T1) && (t<t2+T1))
Fc1 = (Tmax-Tmin)*pow(sin(kn*(t-t1-T1)),2)+Tmin;
if(t>t2+T1)
Fc1 = Tmax;
return Fc1;
}
// Расчёт силы управления
static double Fccc(double V, double Vnom, double Ll, double Lnom, double t)
{
double kl = 0.1; // коэффициент обратной связи по длине троса
double kv = 0.5; // коэффициент обратной связи по скорости троса
double Fcopr = kv*(V - Vnom) + kl*(Ll - Lnom);
return Fcopr;
}

// Расчёт силы упругости для развёрнутого участка
static double Fyprnom(double LL, double Ln)
{
double Fyp ;
double cl = 1000; // жёсткость троса
if(LL-Ln>0)
{Fyp = (cl/Ln)*(LL-Ln);}
else
{Fyp = 0;}
return Fyp;
}

};

A =gY; // Первоначальная инициализация вектора правых частей
double *r = new double[2]; // Радиус-вектора концевых точек
double L; // Длина между концевыми точками

// Определения длины радиус-векторов КА и СК
for(int i = 0; i<=1; i++)
{
r[i] = Forces::ModVect(gY[i][0],gY[i][2], gY[i][4]);
}

// Длина участка троса между точками крепления к КА и СК
L = Forces::ModVect(gY[1][0] - gY[0][0],gY[1][2] - gY[0][2], gY[1][4] - gY[0][4]);

A[0][0] = gY[0][1];
A[0][1] = (-mu/(r[0]*r[0]))*gY[0][0]/r[0] - (Forces::Fypr(L,gY[2][4])*(gY[0][0] -
gY[1][0])/L)/mka;
A[0][2] = gY[0][3];
```

```
A[0][3] = (-mu/(r[0]*r[0]))*gY[0][2]/r[0] - (Forces::Fypr(L,gY[2][4])*(gY[0][2] - gY[1][2])/L)/mka;
A[0][4] = gY[0][5];
A[0][5] = (-mu/(r[0]*r[0]))*gY[0][4]/r[0] - (Forces::Fypr(L,gY[2][4])*(gY[0][4] - gY[1][4])/L)/mka;

A[1][0] = gY[1][1];
A[1][1] = (-mu/(r[1]*r[1]))*gY[1][0]/r[1] + (Forces::Fypr(L,gY[2][4])*(gY[0][0] - gY[1][0])/L)/(mck);
A[1][2] = gY[1][3];
A[1][3] = (-mu/(r[1]*r[1]))*gY[1][2]/r[1] + (Forces::Fypr(L,gY[2][4])*(gY[0][2] - gY[1][2])/L)/(mck);
A[1][4] = gY[1][5];
A[1][5] = (-mu/(r[1]*r[1]))*gY[1][4]/r[1] + (Forces::Fypr(L,gY[2][4])*(gY[0][4] - gY[1][4])/L)/(mck);
A[2][0] = gY[2][1];
A[2][1] = -2*(gY[2][3]/gY[2][2])*(PSI+gY[2][1]) - 1.5*PSI*PSI*sin(2*gY[2][0]);
A[2][2] = gY[2][3];
A[2][3] = gY[2][2]*(pow((gY[2][1]+PSI),2)-pow(PSI,2)*(1-3*cos(gY[2][0])*cos(gY[2][0]))) - Forces::Fc(gY[2][3],gY[2][2],gt)/mck;
A[2][4] = gY[2][5];
A[2][5] = (rkat/Ikat)*(- Forces::Fccc( gY[2][5],gY[2][3],gY[2][4],gY[2][2], gt) + Forces::Fypr(L,gY[2][4]));
```

Исследуя листингу, можно заметить, что он содержит как раз класс, где определены статические функции (класс *Forces*), определяющие силы упругости троса (*Fypr*, *Fc*, *Fccc*, *Fyprnom*). Данные функции используются в определение правых частей (глобальный вектор A). Замечание: сила (*Fyprnom)* здесь не используется, так как она применяется к уже развернутому участку троса и здесь рассматривается процесс развёртывания. Так же в классе *Forces* определена дополнительная функция *(ModVect)* определяющая модуль трёхмерного вектора.

В листинге в самом начале проводиться изменение глобальной переменной *gk,* как описывалось выше, данная переменная необходима для определения предикатов входа в модули расчёта коэффициентов $\left(k_i,\ i=1..4\right)$.

Функции расчёта правых частей для двух участков и для произвольного числа участков приведены в приложении 3.

5.6 Модуль добавления нового участка троса

Процесс интегрирования развёртывания троса сопряжён с проблемой добавления нового участка троса.

Процесс добавления нового участка троса в среде PGraph можно реализовать при помощи механизма обратной связи модуля интегрирования и модуля добавления новой точки соединённого с модулем перерасчёта начальных условий системы ОДУ.

Схема добавления нового участка изображена на рис. 13:

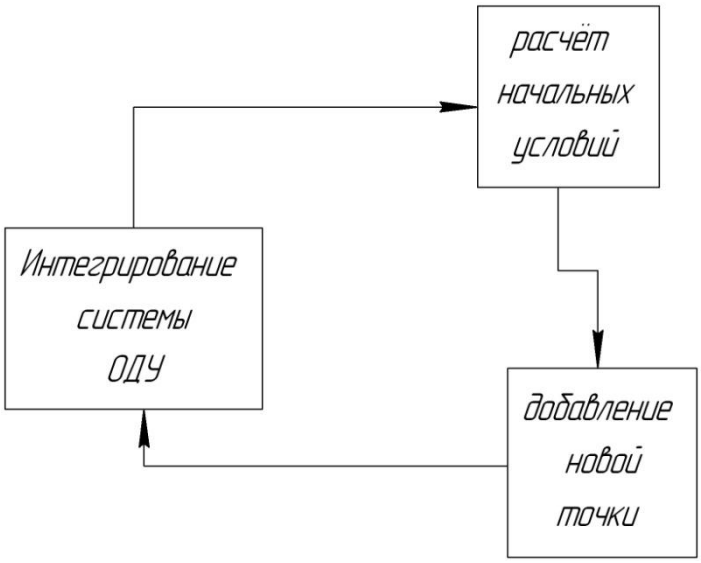


Рис. 13 – Схема для реализации алгоритма добавления новой точки

Из рисунка видно, что расчёт начальных условий производится по данным произведённым на предыдущем этапе интегрирования (интегрирования системы для меньшего числа участков троса). Выход из данного цикла интегрирования расчёта новых начальных условий и добавления уравнений движения новой точки (нового участка троса) осуществляется, когда число участков троса достигнет максимального заданного значения.

Обобщая всё вышесказанное, приведём полную схему расчёта развёртывания КТС, реализованную в среде PGraph, которая выглядит так, как изображено на рис. 14:

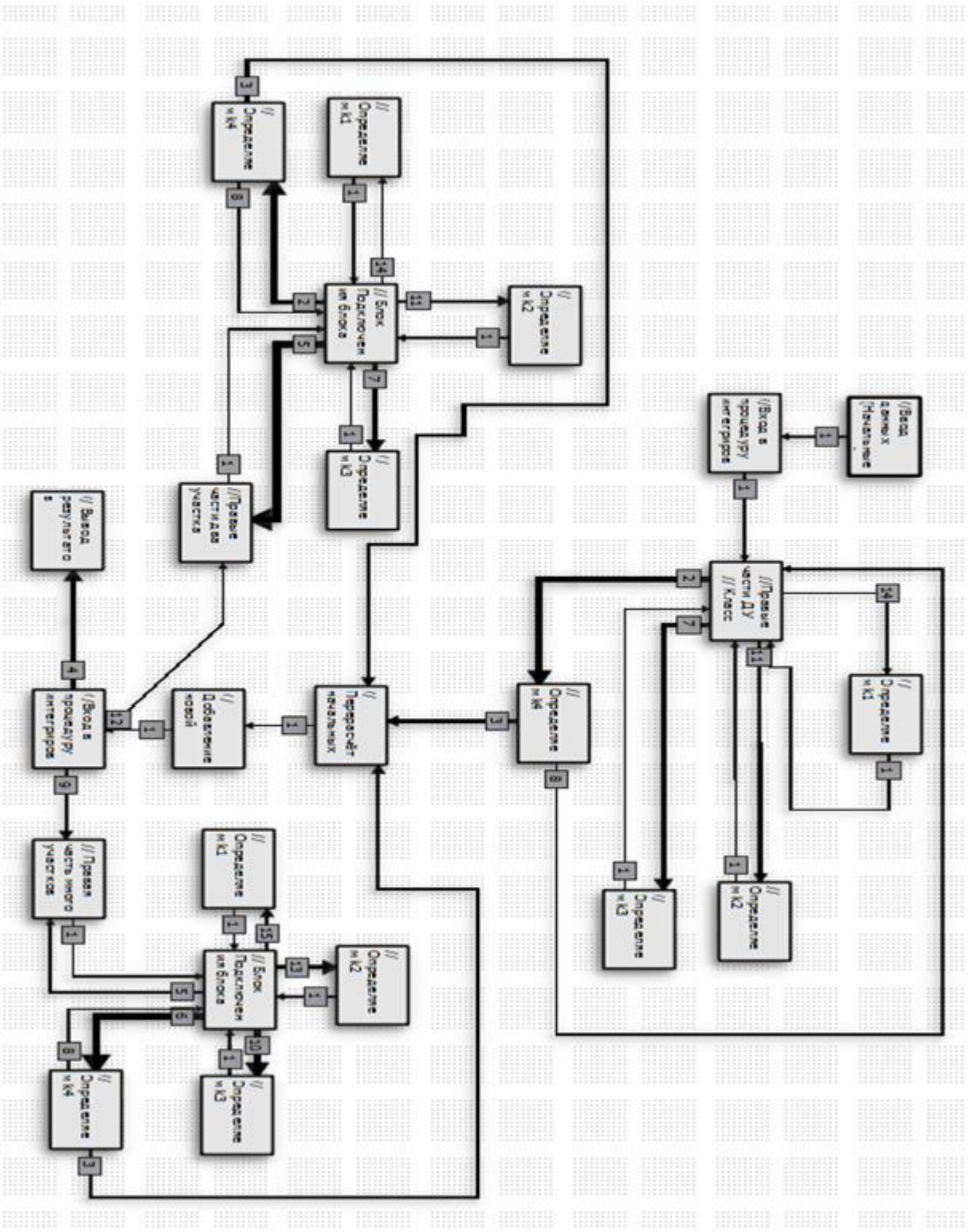


Рис. 14 – Полная схема моделирования процесса развёртывания КТС

Схема, изображённая на рис. 14 реализует весь механизм расчёта процесса развёртывания КТС. Реализация всех акторов и предикатов приведена в Приложение 3.

Список использованных источников

1. Hoyt, R. P. Remediation of radiation belts using electrostatic tether structures [Text] / R.P. Hoyt, B. M. Minor // Aerospace Conf., 2005 IEEE. Big Sky, MT, pp. 583-594.
2. Robert, P. Hoyt. Cislunar tether transport system [Text] / Robert P. Hoyt, Chancey Uphoff // AIAA-99-2690, pp. 1-15. Электронный ресурс, режим доступа свободный: http://www.tethers.com/papers/CislunarAIAAPaper.pdf - Загл. с экрана.
3. Robert L. Forward. Mars-Eath rapid interplanetary tether transport (MERIT) system: I. Initial feasibility analysis [Text] / Robert L. Forward, Gerald D. Nordley // AIAA-99-2151, pp. 1-18. Электронный ресурс, режим доступа свободный: http://www.tethers.com/papers/MERITT.pdf- Загл. с экрана.
4. Hoyt, R. P. The Terminator Tape: A cost-effective de-orbit module for end-of-life disposal of LEO satellites [Text] / Hoyt, R. P., et al. // AIAA Space 2009 Conference and Exposition, September 14, 2009 -September 17, pp. 1 - 9.
5. Белецкий, В. В. Динамика космических тросовых систем [Текст] / В. В. Белецкий, Е. М. Левин. – М.: Наука, 1990. – 336 с.
6. Cosmo, M. L. Tethers in space handbook // Third edition [Text] / M.L. Cosmo, E. C. Lorenzini. - December, 1997, pp. 1-274.
   Проект TSS-1 электронный ресурс, режим доступа свободный: http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=TSS-1 - Загл. с экрана.

7. Tyc, G. On the Dynamics of Spinning Tethered Space Vehicles [Text] / Tyc, G. and Ray P.S. Han // Philosophical Transactions: Mathematical, Physical & Engineering Sciences of the Royal Society, 359, 2161-2190, 2001.
8. Электронный ресурс Канадского космического агентства, режим доступа свободный: http://www.asc-csa.gc.ca/images/recherche/images/90_hr.jpg - Загл. с экрана, язык англ.
9. Michel Kruijff. Qualification and in-flight demonstration of a European tether deployment and momentum transfer system on YES2 [Text] / Michel Kruijff, Eric J. van der Heide, Marco Stelze // Proceedings of the IAA symposium on small satellites system and services, Rhodes, Greese, 26-30 May 2008, pp. 1-17.
10. Michel Kruijff. Appliciability of tether deployment simulation and tests based on YES 2 flight data [Text] / Michel Kruijff. AIAA -2008-7036, pp. 1-24.
11. Williams, P. YES2 optimal trajectories in presence of eccentricity and aerodynamic drag [Text] / P. Williams, A. Hyslop, M. Stelzer, M. Kruijff // IAC -06-02.3.04, Valencia, 2006 and Acta Astronautica. Volume 64, issue 7 – 8, april-may 2009, pp. 745 – 769.
12. Проект YES2 [Электронный ресурс]. Режим доступа свободный: http://www.yes2.info/ - Загл. с экрана., язык англ.
13. Robert, L. Forward. Terminator tether: spacecraft deorbit device [Text] / Robert L. Forward, Robert P. Hoyt // Journal of spacecraft and rockets – Vol. 37, № 2, March-April 2000, pp. 187 – 196.
14. Guang Zhai. System dynamics and feedforward control for tether-net space robot system [Text] / Guang Zhai, Yue Qiu, Bin Liang and Cheng Li // International journal of advanced robotic systems, vol. 6, № 2 (2009) ISSN 1729-8806, pp. 137 – 144.

15. Наумов О. Н. Разработка моделей и анализ пространственного движения спускаемой капсулы относительно центра масс при развёртывании космической тросовой системы: дис. канд. техн. наук : 05.07.09 : защищена 19.10.2012 / Наумов Олег Николаевич; Самар. гос. аэрокосм. ун-т им. С. П. Королева. – Самара : СГАУ, 2012. – 180 с.
16. Шейников И. В. Формирование программного управления развертыванием орбитальных тросовых систем для выполнения транспортных операций с малыми космическими аппаратами [Текст] : дис. канд. техн. наук : 05.07.09 : защищена 27.12.2010 / Шейников Игорь Владимирович ; М-во образования и науки Рос. Федерации, Самар. гос. аэрокосм. ун-т им. С. П. Королева (Нац. исслед. ун-т). - Самара, 2010. - 130 с.
17. Алпатов, А. П. Динамика космических систем с тросовыми и шарнирными соединениями [Текст] / А. П. Алпатов и др. - Москва- Ижевск: НИЦ «Регулярная и хаотическая динамика», Институт компьютерных исследований, 2007. – 560 с.
18. Фефелов Д. И. Моделирование и анализ развертывания и снижения с околоземной орбиты тросовой системы со спускаемой капсулой [Текст] : дис. канд. техн. наук : 05.07.09 : защищена 27.09.2007 / Фефелов Денис Иванович ; Самар. гос. аэрокосм. ун-т им. С. П. Королева. - Самара : СГАУ, 2007. - 131 с.
19. Заболотнов, Ю. М. Движение спускаемой капсулы относительно центра масс при развертывании орбитальной тросовой системы [Текст] / Ю. М. Заболотнов, О. Н. Наумов // Космические исследования, т.50, № 2, 2012. - С.177-187.

20. Наумов, О. Н. Демпфирование колебаний спускаемой капсулы при управляемом развертывании тросовой системы [Текст] / О. Н. Наумов // Полет, №2, 2012. – С. 45-50.
21. Наумов, О. Н. Статистический анализ вращательного движения легкой спускаемой капсулы при развертывании космической тросовой системы [Текст] / О. Н. Наумов // Изв. вузов. Авиационная техника, №2, 2012. - С. 37-40.
22. Заболотнов, Ю. М. Анализ пространственного вращательного движения концевого тела при развертывании орбитальной тросовой системы [Текст] / Ю. М. Заболотнов, О. Н. Наумов // Известия СНЦ РАН. - Т.11. №3. – 2009. - С. 249-256.
23. Наумов, О. Н. Анализ влияния статической и динамической асимметрии на вращательное движение капсулы при управляемом развертывании тросовой систем [Текст] / О. Н. Наумов // Электронный журнал Труды МАИ. Выпуск № 43. – 2011. URL: http://www.mai.ru/science/trudy/published.php?ID=24767 - загл. с экрана.
24. Наумов, О. Н. Статистический анализ вращательного движения капсулы на тросе [электронный ресурс] // О. Н. Наумов. Электронный многопредметный научный журнал «Исследовано в России». – М: МФТИ, 072, стр. 943-952, 2009 г. Режим доступа http://zhurnal.ape.relarn.ru/articles/2009/072.pdf - Загл. с экрана.
25. Мирер С. А. Механика космического полета. Орбитальное движение: Учебное пособие / С. А. Мирер. – М: Резолит, 2007. – 270 с.: ил.
26. Лойцянский, Л. Г. Курс Теоретической механики. В 2 т. / Л. Г. Лойцянский, А. И. Лурье. – М.: Дрофа, 2006.
27. Никитин, Н. Н. Курс теоретической механики: Учебник для машиностроительных и приборостроительных

специальностей вузов / Н. Н. Никитин. – 6-е изд., перераб. И доп. – М.: Высш. шк., 2003. – 719 с.: ил.

28. Маркеев А. П. Теоретическая механика: Учебник для университетов. – Ижевск: НИЦ «Регулярная и хаотичная динамика», 2001, 592 с.
29. Гантмахер Ф. Р. Лекции по аналитической механике: Учебное пособие для вузов / Под ред. Е. С. Пятницкого. – 3-е изд. – М.: ФИЗМАТЛИТ, 2005. – 264 с.
30. Галиуллин А. С. Аналитическая динамика. Учебник по аналитической динамике. – М.: Изд-во РУДН, 1998. – 441 с.
31. Герберт Шилдт. Полный справочник по C++, 4-е издание. Пер. с англ. – М.: Издательский дом “Вильямс”, 2010. – 800 с.: ил.
32. Бьерн Страуструп. Язык программирования C++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2012 г. – 1136 с.: ил.
33. Кузнецов, М. В. C++, мастер-класс в задачах и примерах / М. В. Кузнецов, И. В. Симдянов. – СПб.: БХВ-Петербург, 2007. – 480 с.: ил.
34. Майерс С. Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов: Пер. с англ. – М.: ДМК Пресс; Спб.: Питер, 2006. – 240 с.: ил.
35. Стефан К. Дьюхэрст. Скользкие места C++. Как избежать проблем при проектировании и компиляции ваших программ. – М.: ДМК Пресс, 2006. – 264 с.: ил.
36. Леен Аммераль. STL для программистов на C++. Пер. с англ. – М.: ДМК, 1999 – 240 с., ил.

37. Москвин П. В. Азбука STL. – 2-е изд., стереотип. – М.: Горячая линия – Телеком, 2012. – 262 с.: ил.
38. Дэвид Вандервурд, Николай М. Джосаттис. Шаблоны C++: справочник разработчика. : Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 544 с.: ил.
39. Финогенов К. Г. Win32. Основы программирования. – 2-е изд., испр. и дополн. – М.: ДИАЛОГ-МИФИ. 2006. – 416 с.
40. Щупак Ю. А. Win32 API. Разработка приложений для Windows. – СПб.: Питер, 2008. – 592 с.: ил.
41. Литвиненко Н. А. Технология программирования на C++. Win32 API-приложения. – СПб.: БХВ-Петербург, 2010. – 288 с.: ил.
42. Скиена С. Алгоритмы. Руководство по разработке. – 2-е изд.: Пер. с англ. – СПб.: БХВ – Петербург, 2013. – 720 с.: ил.
43. Пирумов, У. Г. Численные методы: теория и практика : учеб. Пособие для бакалавров / У. Г. Пирумов [и др.]. – 5-е изд., перераб. и доп. – М.: Издательство Юрайт, 2012. – 421 с.
44. Волков Е. А. Численные методы: Учебное пособие. 3-е изд., испр. – СПб.: Издательство «Лань», 2004. – 256 с.
45. Мэтьюз, Джон Г., Финк, Куртис Д. Численные методы. Использование MATLAB, 3-е издание.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2001. – 720 с.: ил.

Приложение 1. Словарь данных

Таблица 1. Данные

№	Данные ПО	Тип	Начальное значение	Комментарий
1	A	DoubleDvector	-	Матрица правых частей
2	Ikat	double	1	Момент инерции Механизма управления
3	Lsum	double	0	Суммарная длина троса
4	Morb	DoubleDvector		Матрица перехода От геоцентрической к орбитальной СК
5	PI	double	3.14159265358	Константа Пи
6	PSI	double	0.00115872474	Угловая скорость движения спутника на круговой орбите
7	VecNachYsl	DoubleDvector		Вектор начальных условий
8	c	double	1000	Жёсткость троса
9	gH	double	300000	Начальная высота орбиты
10	gI	double	0.5	Угол наклона орбиты
11	gIMatrOrb	DoubleDvector		Матрица поворота
12	gIstAnom	double	0.2	Угол истинной аномалии
13	gKbool	int	0	Вспомогательная переменная, необходима для определения предикатов
14	gLmax	double	100	Полная длина троса

Таблица 1. Продолжение

№	Данные ПО	Тип	Начальное значение	Комментарий
15	gLnom	double	100	Номинальная длина участка
16	gLrealYch	double	0	Переменная длина участка
17	gNych	int	1	Число участков троса
18	gTeta	double	0.7	Долгота восходящего узла
19	gTetaMatrOrb	DoubleDvector		Матрица перехода
20	gU	double	1	Угол поворота в плоскости орбиты
21	gUMatrOrb	DoubleDVector		Матрица перехода
22	gVnotstr	double	1.98	Скорость отстрела СК
23	gY	DoubleDvector		Вектор значений решения ДУ
24	gdr	double	0.05	Начальная длина Первого участка
25	ge	double	0.01	Величина эксцентрисетета орбиты
26	gk	int	0	Определяет номер k
27	grck	doublevector		Начальные значения вектора СК
28	grka	doublevector		Начальные значения вектора КА
29	gt	double	0	Глобальное время
30	gvka	doublevector		Начальное значение скорости КА
31	gvck	doublevector		Начальное значение скорости

				СК

Типы данных:

Стандартные типы данных: int – целочисленный тип, double – вещественный тип.

Пользовательские типы данных:

typedef std::vector<double> doublevector;

typedef std::vector<std::vector<double> > DoubleDvector;

Приложение 2. Содержание заголовочного файла с определением перегруженных операций

```
#include <math.h>
#include <iostream>
#include <vector>
#include <list>
#include <memory>
#include <windows.h>
using namespace std;

inline double ModVect(double x, double y, double z)
	{
	double r = sqrt(x*x + y*y + z*z);
	return r;
	}

// Функция показывающая содержимое вектора
inline void showVector(vector<double> V)
{
	for(int i = 0; i<V.size(); i++)
	{
	cout<<V[i]<<" ";
	}
	cout<<endl;
}

// Функция показывающая содержимое двойного вектора
inline void showVector(vector<vector<double> > V)
{
	for(int i = 0; i<V.size(); i++)
	{
		for(int j = 0; j<V[i].size(); j++)
			cout<<V[i][j]<<" ";
	cout<<endl;
	}
```

```
}

//Сумма двух векторов
inline vector<double> operator+(vector<double> const &V1, vector<double>const & V2)
{
	int except;
	size_t vsize = V1.size();
	vector<double> Vsum(vsize,0);
	try
	{
	  if(V1.size() != V2.size())
	  {
	   throw new vector<double>;
	  }
	  else
	  {
	  //Расчёт суммы двух векторов

	  for(int i = 0; i<vsize; i++)
	    {
		Vsum[i] = V1[i] + V2[i];
	    }
	  }
	}
	catch(vector<double>*)
	{
	  cout<<"Error operator+ (type: vector<double>)"<<endl;
	  exit(1);
	}
	return Vsum;
}
//Разность двух векторов
inline vector<double> operator-(vector<double> const &V1, vector<double>const & V2)
{
	int except;
	size_t vsize = V1.size();
	vector<double> Vsum(vsize,0);
	try
	{
	  if(V1.size() != V2.size())
	  {
	   throw new vector<double>;
	  }
	  else
	  {
	  //Расчёт суммы двух векторов
	  for(int i = 0; i<vsize; i++)
	    {
		Vsum[i] = V1[i] - V2[i];
	    }
	  }
	}
```

```
        catch(vector<double>*)
        {
          cout<<"Error operator- (type: vector<double>)"<<endl;
          exit(1);
        }
        return Vsum;
}
//Произведение вектора на число
inline vector<double> operator*(double num, vector<double> const& Vect1)
{
        vector<double> VectT1(Vect1.size());
        for(int i = 0; i<Vect1.size(); i++)
        {
        VectT1[i] = num*Vect1[i];
        }
        return VectT1;
}
inline vector<double> operator*(vector<double> const& Vect1, double num)
{
        vector<double> VectT1(Vect1.size());
        for(int i = 0; i<Vect1.size(); i++)
        {
        VectT1[i] =Vect1[i]*num;
        }
        return VectT1;
}

// Деление вектора на число
inline vector<double> operator/(vector<double> const& Vect1, double num)
{
        vector<double> VectT1(Vect1.size());
        for(int i = 0; i<Vect1.size(); i++)
        {
        VectT1[i] =Vect1[i]/(num);
        }
        return VectT1;
}

// Сумма двух двумерных векторов
inline vector<vector<double> > operator+(vector<vector<double> > const &VDouble1,
vector<vector<double> > const &VDouble2 )
{
       size_t vsizestrok = VDouble1.size();
        size_t vsizestolb = VDouble1[0].size();
         vector<vector<double> > Vsum(VDouble1);
         auto_ptr<vector<vector<double> > > p1(new vector<vector<double> >);
          try
          {
          if(VDouble1.size() != VDouble2.size() || VDouble1[0].size() != VDouble2[0].size())
//Обрабатываем исключение
                throw p1;
          else //Проводим расчёт суммы двумерных массивов
```

```
            {
            for(int i = 0; i<vsizestrok; i++)
                    {
                    for(int j = 0; j<vsizestolb; j++)
                      {
                      Vsum[i][j] = VDouble1[i][j] + VDouble2[i][j];
                      }
                    }//конец внешнего цикла for
             }
          }// Конец try
          catch(auto_ptr<vector<vector<double> > >)
          {
             cout<<"Error operator+ (type: vector<vector<double> >)"<<endl;
             exit(1);
          }

return Vsum;
}

//Разность двух двумерных векторов
inline vector<vector<double> > operator-(vector<vector<double> > const &VDouble1,
vector<vector<double> > const &VDouble2 )
{
          size_t vsizestrok = VDouble1.size();
          size_t vsizestolb = VDouble1[0].size();
          vector<vector<double> > Vrazn(VDouble1);
          auto_ptr<vector<vector<double> > > p1(new vector<vector<double> >);
           try
           {
          if(VDouble1.size() != VDouble2.size() || VDouble1[0].size() != VDouble2[0].size())
//Обрабатываем исключение
                    throw p1;
          else //Проводим расчёт суммы двумерных массивов
             {
             for(int i = 0; i<vsizestrok; i++)
                    {
                    for(int j = 0; j<vsizestolb; j++)
                      {
                      Vrazn[i][j] = VDouble1[i][j] - VDouble2[i][j];
                      }
                    }//конец внешнего цикла for
              }
          }// Конец try
          catch(auto_ptr<vector<vector<double> > >)
          {
            cout<<"Error operator- (type: vector<vector<double> >)"<<endl;
            exit(1);
          }
return Vrazn;
}

// Произведение числа на двумерный вектор
```

```
inline vector<vector<double> > operator*(double const &number, vector<vector<double> > const &VDouble1 )
{
        size_t vsizestrok = VDouble1.size();
         size_t vsizestolb = VDouble1[0].size();
         vector<vector<double> > Vrez(VDouble1);
         for(int i = 0; i<vsizestrok; i++)
           {
           for(int j = 0; j<vsizestolb; j++)
                    {
                    Vrez[i][j] = number*VDouble1[i][j];
                    }
           }//конец внешнего цикла for
return Vrez;
}

// Оператор реализующий произведения матриц
inline vector<vector<double> > operator*(vector<vector<double> > Vec1, vector<vector<double> > Vec2 )
{
        vector<vector<double> > Vect(Vec1);
        for(int i=0; i<Vec1.size(); i++)
        {
          for(int j = 0; j<Vec1[0].size(); j++)
          {
          Vect[i][j]= Vec1[i][0]*Vec2[0][j]+Vec1[i][1]*Vec2[1][j]+Vec1[i][2]*Vec2[2][j];
          }
        }
        return Vect;
}

// Оператор реализующий произведение матрицы на вектор
inline vector<double> operator*(vector<vector<double> > Matriza, vector<double>  Vector )
{
        vector<double> VectRez(Vector.size(),0);
        for(int i = 0; i<Matriza.size(); i++)
        {
          for(int j = 0; j<Vector.size(); j++)
          {
          VectRez[i] += Matriza[i][j]*Vector[j];
          }
        }
return VectRez;
}
```

Приложение 3. Акторы и предикаты

```
// Поворот на угол gI
int nsizeMatr = 3;
int msizeMatr = 3;
gIMatrOrb.resize(nsizeMatr);

for(int i = 0; i<nsizeMatr; i++)
gIMatrOrb[i].resize(msizeMatr);

gIMatrOrb[0][0] = 1; gIMatrOrb[0][1] = 0; gIMatrOrb[0][2] = 0;
gIMatrOrb[1][0] = 0; gIMatrOrb[1][1] = cos(gI); gIMatrOrb[1][2] = sin(gI);
gIMatrOrb[2][0] = 0 ; gIMatrOrb[2][1] = -sin(gI) ; gIMatrOrb[2][2] = cos(gI);
```

```
// Поворот на угол gU
int nsizeMatr = 3;
int msizeMatr = 3;
gUMatrOrb.resize(nsizeMatr);

for(int i = 0 ; i<nsizeMatr; i++)
gUMatrOrb[i].resize(msizeMatr);

gUMatrOrb[0][0] = cos(gU); gUMatrOrb[0][1] = sin(gU); gUMatrOrb[0][2] = 0;
gUMatrOrb[1][0] = -sin(gU); gUMatrOrb[1][1] = cos(gU); gUMatrOrb[1][2] = 0;
gUMatrOrb[2][0] = 0 ; gUMatrOrb[2][1] = 0 ; gUMatrOrb[2][2] = 1;
```

```
// Поворот на угол восходящего узла
int nsizeMatr = 3;
int msizeMatr = 3;
gTetaMatrOrb.resize(nsizeMatr);

for(int i=0; i<nsizeMatr; i++)
gTetaMatrOrb[i].resize(msizeMatr);

gTetaMatrOrb[0][0] = cos(gTeta); gTetaMatrOrb[0][1] = sin(gTeta); gTetaMatrOrb[0][2] = 0;
gTetaMatrOrb[1][0] = -sin(gTeta); gTetaMatrOrb[1][1] = cos(gTeta); gTetaMatrOrb[1][2] = 0;
gTetaMatrOrb[2][0] = 0 ; gTetaMatrOrb[2][1] = 0 ; gTetaMatrOrb[2][2] = 1;
```

```
// Модуль транспонирования матрицы перехода от орбитальной к геоцентрической
int nsizeMatr = Morb.size();
int msizeMatr = Morb[nsizeMatr-1].size();
vector<vector<double> > VectTemp(Morb);

for(int i = 0; i< nsizeMatr; i++)
{
for(int j = 0; j < msizeMatr; j++)
```

```
{
cout<<Morb[i][j]<<" ";
}
cout<<endl;
}

for(int i = 0; i< nsizeMatr; i++)
{
for(int j = 0; j < msizeMatr; j++)
{
VectTemp[i][j] = Morb[j][i];
}
}

Morb = VectTemp;

for(int i = 0; i< nsizeMatr; i++)
{
for(int j = 0; j < msizeMatr; j++)
{
cout<<Morb[i][j]<<" ";
}
cout<<endl;
}
```

```
//Ввод данных (Начальные условия)
cout<<"vvedite polnuu dlinu trosa"<<endl;
double Lmaxl;
cin>>Lmaxl;
gLmax=Lmaxl;
cout<<"vvedite chislo ychastkov trosa"<<endl;
double gNychl;
cin>>gNychl;
gNych = gNychl;
gLnom = gLmax/gNych;

vector<double> V1(6), V2(6), V3(6);
V1[0] = 5626959.160627306; V1[1] = -3909.2288987826587; V1[2] = 2534771.1279602735;
V1[3] = 3229.1513561197344; V1[4] = 2339287.4841479897; V1[5] = 5942.2215045450785;
V2[0] = 5626959.152101611; V2[1] = -3910.9169865308468; V2[2] = 2534771.1241197116;
V2[3] = 3228.3909247813463; V2[4] = 2339287.480603615; V2[5] = 5941.5197182998345;
V3[0] = 0; V3[1] = 0; V3[2] = 0.01; V3[3] = 1.9800; V3[4] = 0.01; V3[5] = 1.98;
VecNachYsl.push_back(V1); VecNachYsl.push_back(V2); VecNachYsl.push_back(V3);
// Добавление новой точки
vector<vector<double> >::iterator itDVector = gY.begin();
gY.insert(itDVector+1, newPointData);
showVector(gY);
```

```
// Перерасчёт начальных условий

class VectorWork
{
```

```
public:
// Скалярное произведение векторов
static double Scal(double x, double y, double z, double Vx, double Vy, double Vz)
{
double rez;
rez = x*Vx+y*Vy+z*Vz;
return rez;
}

// Модуль вектора

static double Modul(double x, double y, double z)
{
double rez = sqrt(x*x+y*y+z*z);
return rez;
}

};

vector<double> dR1, dV1, dR,V1p, V1n, Vn, dV;
vector<double> R1(3), R0(3), V1(3),V0(3);
vector<double> R;
vector<double> V;
vector<double> RV;
//Определение векторов
R0[0] = gY[0][0]; R0[1] = gY[0][2]; R0[2] = gY[0][4];
V0[0] = gY[0][1]; V0[1] = gY[0][3]; V0[2] = gY[0][5];
R1[0] = gY[1][0]; R1[1] = gY[1][2]; R1[2] = gY[1][4];
V1[0] = gY[1][1]; V1[1] = gY[1][3]; V1[2] = gY[1][5];

//Определение начальных условий
dR1 = R1 - R0;
dV1 = V1 - V0;
dR = (1 - gLnom/ VectorWork::Modul(dR1[0],dR1[1],dR1[2]))*dR1;
double cosa =
VectorWork::Scal(dR1[0],dR1[1],dR1[2],dV1[0],dV1[1],dV1[2])/(VectorWork::Modul(dR1[0],
dR1[1],dR1[2])*VectorWork::Modul(dV1[0],dV1[1],dV1[2]));
V1p
=VectorWork::Modul(dV1[0],dV1[1],dV1[2])*cosa*dR1/VectorWork::Modul(dR1[0],dR1[1],d
R1[2]);
V1n = dV1-V1p;
Vn =
(VectorWork::Modul(dR[0],dR[1],dR[2])/VectorWork::Modul(dR1[0],dR1[1],dR1[2]))*V1n;
dV = Vn + V1p;
V = V0+dV;
R = R0+dR;

//Определение вектора начальных значений для новой точки
RV.push_back(R[0]);
RV.push_back(V[0]);
RV.push_back(R[1]);
RV.push_back(V[1]);
```

```
RV.push_back(R[2]);
RV.push_back(V[2]);
newPointData = RV;
cout<<"New Initial value"<<endl;
showVector(newPointData);
```

```
//Модуль перерасчёта начальных условий
double Vn, Vr; // Трансверсальная и радиальная составляющие скорости
double p; // Фокальный параметр
double rz = 6300000; // Радиус Земли
double rorb = rz+gH; // Высота орбиты
vector<double> rka0(3), rck0(3),vka0(3), vck0(3);

// Определяем параметры движения в орбитальной СК
rka0[0] = rorb ; rka0[1] = 0; rka0[2] = 0;
rck0[0] = rorb-gdr; rck0[1] = 0; rck0[2] = 0;
p = rorb*(1+ge*cos(gIstAnom));
cout<<"p= "<<p<<endl;
Vr = sqrt(mu/p)*ge*sin(gIstAnom);
Vn= sqrt(mu/p)*(1+ge*cos(gIstAnom));
vka0[0] = Vr ; vka0[1] = Vn; vka0[2] = 0;
vck0[0] = Vr-gVnotstr; vck0[1] = Vn; vck0[2] = 0;

// Определяем параметры движения в геоцентрической СК
grka = Morb*rka0; grck = Morb*rck0;
gvka = Morb*vka0; gvck = Morb*vck0;

cout<<"grka = "<<endl;
for(int i = 0; i<grka.size(); i++)
cout<<grka[i]<<" ";

cout<<"\n gvka = "<<endl;
for(int i = 0; i<gvka.size(); i++)
cout<<gvka[i]<<" ";

cout<<"grck = "<<endl;
for(int i = 0; i<grck.size(); i++)
cout<<grck[i]<<" ";

cout<<"\n gvck = "<<endl;
for(int i = 0; i<gvck.size(); i++)
cout<<gvck[i]<<" ";
```

```
// Блок Подключения блока интегрирования
cout<<"Internal blok"<<endl;
cout<<"gk ="<<gk<<endl;
```

```
//Вход в процедуру интегрирования два участка и более участков
cout<<"Rachet sledyuhego ychastka"<<endl;
cout<<"Vvedite chifry"<<endl;
```

```
int dalee;
cin>>dalee;
showVector(gY);
gSizePoint = gY.size()-2;
gk=0;
```

```
//Определяем k1
cout<<"Internal K1, gY initial: "<<endl;
//showVector(gY);
k1=A;
gY = gY+ (step/2.0)*k1;
gt = gt + (step/2.0);
gKbool = 1;
```

```
// Определяем k2
k2 = A;
gY = gY - (step/2.0)*k1;
cout<<"Internal K2, gY initial: "<<endl;
//showVector(gY);
gY = gY +(step/2.0)*k2;
gKbool = 1;
```

```
// Определяем k3
k3 = A;
gY = gY - (step/2.0)*k2;
cout<<"Internal K3, gY initial: "<<endl;
//showVector(gY);
gY = gY +(step)*k3;
gt = gt+ (step/2.0);
gKbool = 1;
```

```
// Определяем k4
class Forces
{
public:
static double ModVect(double x, double y, double z)
{
double r = sqrt(x*x + y*y + z*z);
return r;
}
// Расчёт силы упругости
static double Fypr(double Ll, double fkat)
{
double Fy;
double rkatl = 1.0;
double cl = 1000;
if ( (Ll-fkat*rkatl)>0)
{Fy = cl*(Ll-fkat*rkatl)/(fkat*rkatl);}
else
{Fy = 0;}
return Fy;
}
```

```
static double Fc(double V, double Ll, double t)
{
double a = 4.6, b = 3.5, cc = 1.6, Lk = 3000;
double mckl = 20;
double PSIl = 0.00115872474917774;
double PIl = 3.141592653589793;
double Fcopr = mckl*PSIl*PSIl*(a*Ll+b*(V/PSIl)-cc*Lk);
double kn = 0.002, tp = 1700, Tmin = 0.002, Tmax = 1.75, T1 = 6000, kp = 0.002, Tkl = 0.243;
double t1 = tp - PIl/(4*kp);
double t2 = tp + PIl/(4*kn);
double t2p = T1 + PIl/(4*kp);
double t1p = T1 - PIl/(4*kp);
double Fc1;
if(t<t1p)
Fc1 = Fcopr;
if((t>t1p) && (t<t2p))
Fc1 = (Tmin - Tkl)*pow(sin(kp*(t-t1p)),2) + Tkl;
if((t>t2p) && (t<t1+T1))
Fc1 = Tmin;
if((t>t1+T1) && (t<t2+T1))
Fc1 = (Tmax-Tmin)*pow(sin(kn*(t-t1-T1)),2)+Tmin;
if(t>t2+T1)
Fc1 = Tmax;
return Fc1;
}

static double Fccc(double V, double Vnom, double Ll, double Lnom, double t)
{
double kl = 0.1;
double kv = 0.5;
double Fcopr = kv*(V - Vnom) + kl*(Ll - Lnom);
return Fcopr;
}

static double Fyprnom(double LL, double Ln)
{
double Fyp ;
double cl = 1000;
if(LL-Ln>0)
{Fyp = (cl/Ln)*(LL-Ln);}
else
{Fyp = 0;}
return Fyp;
}

};

k4 = A;
gY = gY - (step)*k3;
cout<<"Internal K4, gY initial: "<<endl;
//showVector(gY);
```

```
gY = gY + (step/6.0)*(k1+k2+k3+k4);
showVector(gY);
gLrealYch=Forces::ModVect(gY[1][0] - gY[0][0], gY[1][2] - gY[0][2], gY[1][4] - gY[0][4]);
cout<<"Lreal = "<<gLrealYch<<endl;
cout<<"Fypr = "<<Forces::Fypr(gLrealYch,gY[2][4])<<endl;
int n = gY.size();
if(n>3)
//cout<<"Fypr ="<<Forces::Fypr(Lsum,gY[n-1][4])<<endl;
cout<<"Lsum ="<<Lsum<<" "<<"gY[n-1][4] = "<<gY[n-1][4]<<endl;
gKbool = 1;
```

```
//Правые части два участка
// Класс для расчёта силы упругости и управления
cout<<"internal PravChast"<<"GK ="<<gk<<endl;
gKbool=0;
gk++;
if(gk>4)
gk= 1;

class Forces
{
public:
static double ModVect(double x, double y, double z)
{
double r = sqrt(x*x + y*y + z*z);
return r;
}
// Расчёт силы упругости
static double Fypr(double Ll, double fkat)
{
double Fy;
double rkatl = 1.0;
double cl = 1000;
if ( (Ll-fkat*rkatl)>0)
{Fy = cl*(Ll-fkat*rkatl)/(fkat*rkatl);}
else
{Fy = 0;}
return Fy;
}

static double Fc(double V, double Ll, double t)
{
double a = 4.6, b = 3.5, cc = 1.6, Lk = 3000;
double mckl = 20;
double PSIl = 0.00115872474917774;
double PIl = 3.141592653589793;
double Fcopr = mckl*PSIl*PSIl*(a*Ll+b*(V/PSIl)-cc*Lk);
double kn = 0.002, tp = 1700, Tmin = 0.002, Tmax = 1.75, T1 = 6000, kp = 0.002, Tkl = 0.243;
double t1 = tp - PIl/(4*kp);
double t2 = tp + PIl/(4*kn);
double t2p = T1 + PIl/(4*kp);
double t1p = T1 - PIl/(4*kp);
```

```
double Fc1;
if(t<t1p)
Fc1 = Fcopr;
if((t>t1p) && (t<t2p))
Fc1 = (Tmin - Tkl)*pow(sin(kp*(t-t1p)),2) + Tkl;
if((t>t2p) && (t<t1+T1))
Fc1 = Tmin;
if((t>t1+T1) && (t<t2+T1))
Fc1 = (Tmax-Tmin)*pow(sin(kn*(t-t1-T1)),2)+Tmin;
if(t>t2+T1)
Fc1 = Tmax;
return Fc1;
}

static double Fccc(double V, double Vnom, double Ll, double Lnom, double t)
{
double kl = 0.1;
double kv = 0.5;
double Fcopr = kv*(V - Vnom) + kl*(Ll - Lnom);
return Fcopr;
}

static double Fyprnom(double LL, double Ln)
{
double Fyp ;
double cl = 1000;
if(LL-Ln>0)
{Fyp = (cl/Ln)*(LL-Ln);}
else
{Fyp = 0;}
return Fyp;
}

};


A =gY; // Первоначальная инициализация вектора правых частей
int n =gY.size();
double *r = new double[n-1]; // Радиус-вектора концевых точек
double *L = new double[n-1]; // Длина между концевыми точками
double mtrl = mtr/2; // Присоединённая масса

// Определения длины радиус-векторов
for(int i = 0; i<n-1; i++)
{
r[i] = Forces::ModVect(gY[i][0],gY[i][2], gY[i][4]);
}

// Длина участка троса между точками
for(int i = 0; i<n-2; i++)
{
L[i] = Forces::ModVect(gY[i+1][0] - gY[i][0],gY[i+1][2] - gY[i][2], gY[i+1][4] - gY[i][4]);
```

```
}
Lsum = 0; // Длина развернувшейся части
for(int i = 0; i<n-2; i++)
{Lsum = Lsum+L[i];}
// Расстояние между КА и СК
double Lkask = Forces::ModVect(gY[2][0] - gY[0][0], gY[2][2] - gY[0][2], gY[2][4] - gY[0][4]);
//Правые части Диф. уравнений
A[0][0] = gY[0][1];
A[0][1] = (-mu/(r[0]*r[0]))*gY[0][0]/r[0] - (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][0] - gY[1][0])/L[0])/mka;
A[0][2] = gY[0][3];

A[0][3] = (-mu/(r[0]*r[0]))*gY[0][2]/r[0] - (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][2] - gY[1][2])/L[0])/mka;
A[0][4] = gY[0][5];
A[0][5] = (-mu/(r[0]*r[0]))*gY[0][4]/r[0] - (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][4] - gY[1][4])/L[0])/mka;

A[1][0] = gY[1][1];
A[1][1] = (-mu/(r[1]*r[1]))*gY[1][0]/r[1] + (Forces:: Fypr(Lsum,gY[n-1][4])*(gY[0][0] - gY[1][0])/L[0] - Forces::Fyprnom(L[1], gLnom)*(gY[1][0] - gY[2][0])/L[1])/(mtrl/2);
A[1][2] = gY[1][3];
A[1][3] = (-mu/(r[1]*r[1]))*gY[1][2]/r[1] + ( Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][2] - gY[1][2])/L[0] - Forces::Fyprnom(L[1], gLnom)*(gY[1][2] - gY[2][2])/L[1])/(mtrl/2);
A[1][4] = gY[1][5];
A[1][5] = (-mu/(r[1]*r[1]))*gY[1][4]/r[1] + ( Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][4] - gY[1][4])/L[0] - Forces::Fyprnom(L[1],gLnom)*(gY[1][4] - gY[2][4])/L[1])/(mtrl/2);


A[2][0] = gY[2][1];
A[2][1] = (-mu/(r[2]*r[2]))*gY[2][0]/r[2] + (Forces::Fyprnom(L[1], gLnom)*(gY[1][0] - gY[2][0])/L[1])/(mck+mtrl/2);
A[2][2] = gY[2][3];
A[2][3] = (-mu/(r[2]*r[2]))*gY[2][2]/r[2] + (Forces::Fyprnom(L[1], gLnom)*(gY[1][2] - gY[2][2])/L[1])/(mck+mtrl/2);
A[2][4] = gY[2][5];
A[2][5] = (-mu/(r[2]*r[2]))*gY[2][4]/r[2] + (Forces::Fyprnom(L[1], gLnom)*(gY[1][4] - gY[2][4])/L[1])/(mck+mtrl/2);
A[3][0] = gY[3][1];
A[3][1] = -2*(gY[3][3]/gY[3][2])*(PSI+gY[3][1]) - 1.5*PSI*PSI*sin(2*gY[3][0]);
A[3][2] = gY[3][3];
A[3][3] = gY[3][2]*(pow((gY[3][1]+PSI),2)-pow(PSI,2)*(1-3*cos(gY[3][0])*cos(gY[3][0]))) - Forces::Fc(gY[3][3],gY[3][2],gt)/mck;
A[3][4] = gY[3][5];
A[3][5] = (rkat/Ikat)*(-Forces::Fccc(gY[3][5],gY[3][3],gY[3][4],gY[3][2], gt) + Forces::Fypr(Lsum,gY[n-1][4]));
delete []r;
delete []L;
```

// Правые части ДУ один участок

```
// Класс для расчёта силы упругости и управления
gk++;
if(gk>4)
gk= 1;
class Forces
{
public:
static double ModVect(double x, double y, double z)
{
double r = sqrt(x*x + y*y + z*z);
return r;
}
// Расчёт силы упругости
static double Fypr(double Ll, double fkat)
{
double Fy;
double rkatl = 1.0;
double cl = 1000;
if ( (Ll-fkat*rkatl)>0)
{Fy = cl*(Ll-fkat*rkatl)/(fkat*rkatl);}
else
{Fy = 0;}
return Fy;
}

static double Fc(double V, double Ll, double t)
{
double a = 4.6, b = 3.5, cc = 1.6, Lk = 3000;
double mckl = 20;
double PSIl = 0.00115872474917774;
double PIl = 3.141592653589793;
double Fcopr = mckl*PSIl*PSIl*(a*Ll+b*(V/PSIl)-cc*Lk);
double kn = 0.002, tp = 1700, Tmin = 0.002, Tmax = 1.75, T1 = 6000, kp = 0.002, Tkl = 0.243;
double t1 = tp - PIl/(4*kp);
double t2 = tp + PIl/(4*kn);
double t2p = T1 + PIl/(4*kp);
double t1p = T1 - PIl/(4*kp);
double Fc1;
if(t<t1p)
Fc1 = Fcopr;
if((t>t1p) && (t<t2p))
Fc1 = (Tmin - Tkl)*pow(sin(kp*(t-t1p)),2) + Tkl;
if((t>t2p) && (t<t1+T1))
Fc1 = Tmin;
if((t>t1+T1) && (t<t2+T1))
Fc1 = (Tmax-Tmin)*pow(sin(kn*(t-t1-T1)),2)+Tmin;
if(t>t2+T1)
Fc1 = Tmax;
return Fc1;
}

static double Fccc(double V, double Vnom, double Ll, double Lnom, double t)
```

```
{
double kl = 0.1;
double kv = 0.5;
double Fcopr = kv*(V - Vnom) + kl*(Ll - Lnom);
return Fcopr;
}

static double Fyprnom(double LL, double Ln)
{
double Fyp ;
double cl = 1000;
if(LL-Ln>0)
{Fyp = (cl/Ln)*(LL-Ln);}
else
{Fyp = 0;}
return Fyp;
}

};


A =gY; // Первоначальная инициализация вектора правых частей
double *r = new double[2]; // Радиус-вектора концевых точек
double L; // Длина между концевыми точками

// Определения длины радиус-векторов
for(int i = 0; i<=1; i++)
{
r[i] = Forces::ModVect(gY[i][0],gY[i][2], gY[i][4]);
}

// Длина участка троса между точками
L = Forces::ModVect(gY[1][0] - gY[0][0],gY[1][2] - gY[0][2], gY[1][4] - gY[0][4]);

A[0][0] = gY[0][1];
A[0][1] = (-mu/(r[0]*r[0]))*gY[0][0]/r[0] - (Forces::Fypr(L,gY[2][4])*(gY[0][0] - gY[1][0])/L)/mka;
A[0][2] = gY[0][3];
A[0][3] = (-mu/(r[0]*r[0]))*gY[0][2]/r[0] - (Forces::Fypr(L,gY[2][4])*(gY[0][2] - gY[1][2])/L)/mka;
A[0][4] = gY[0][5];
A[0][5] = (-mu/(r[0]*r[0]))*gY[0][4]/r[0] - (Forces::Fypr(L,gY[2][4])*(gY[0][4] - gY[1][4])/L)/mka;

A[1][0] = gY[1][1];
A[1][1] = (-mu/(r[1]*r[1]))*gY[1][0]/r[1] + (Forces::Fypr(L,gY[2][4])*(gY[0][0] - gY[1][0])/L)/(mck);
A[1][2] = gY[1][3];
A[1][3] = (-mu/(r[1]*r[1]))*gY[1][2]/r[1] + (Forces::Fypr(L,gY[2][4])*(gY[0][2] - gY[1][2])/L)/(mck);
A[1][4] = gY[1][5];
```

```
A[1][5] = (-mu/(r[1]*r[1]))*gY[1][4]/r[1] + (Forces::Fypr(L,gY[2][4])*(gY[0][4] - gY[1][4])/L)/(mck);
A[2][0] = gY[2][1];
A[2][1] = -2*(gY[2][3]/gY[2][2])*(PSI+gY[2][1]) - 1.5*PSI*PSI*sin(2*gY[2][0]);
A[2][2] = gY[2][3];
A[2][3] = gY[2][2]*(pow((gY[2][1]+PSI),2)-pow(PSI,2)*(1-3*cos(gY[2][0])*cos(gY[2][0]))) - Forces::Fc(gY[2][3],gY[2][2],gt)/mck;
A[2][4] = gY[2][5];
A[2][5] = (rkat/Ikat)*(- Forces::Fccc( gY[2][5],gY[2][3],gY[2][4],gY[2][2], gt) + Forces::Fypr(L,gY[2][4]));
delete []r;
```

```
// Правая часть много участков троса
cout<<"internal PravChast Mnogo"<<"GK ="<<gk<<endl;
gKbool=0;
gk++;
if(gk>4)
gk= 1;

class Forces
{
public:
static double ModVect(double x, double y, double z)
{
double r = sqrt(x*x + y*y + z*z);
return r;
}
// Расчёт силы упругости
static double Fypr(double Ll, double fkat)
{
double Fy;
double rkatl = 1.0;
double cl = 1000;
if ( (Ll-fkat*rkatl)>0)
{Fy = cl*(Ll-fkat*rkatl)/(fkat*rkatl);}
else
{Fy = 0;}
return Fy;
}

static double Fc(double V, double Ll, double t)
{
double a = 4.6, b = 3.5, cc = 1.6, Lk = 3000;
double mckl = 20;
double PSIl = 0.00115872474917774;
double PIl = 3.141592653589793;
double Fcopr = mckl*PSIl*PSIl*(a*Ll+b*(V/PSIl)-cc*Lk);
double kn = 0.002, tp = 1700, Tmin = 0.002, Tmax = 1.75, T1 = 6000, kp = 0.002, Tkl = 0.243;
double t1 = tp - PIl/(4*kp);
double t2 = tp + PIl/(4*kn);
double t2p = T1 + PIl/(4*kp);
```

```
double t1p = T1 - PIl/(4*kp);
double Fc1;
if(t<t1p)
Fc1 = Fcopr;
if((t>t1p) && (t<t2p))
Fc1 = (Tmin - Tkl)*pow(sin(kp*(t-t1p)),2) + Tkl;
if((t>t2p) && (t<t1+T1))
Fc1 = Tmin;
if((t>t1+T1) && (t<t2+T1))
Fc1 = (Tmax-Tmin)*pow(sin(kn*(t-t1-T1)),2)+Tmin;
if(t>t2+T1)
Fc1 = Tmax;
return Fc1;
}

static double Fccc(double V, double Vnom, double Ll, double Lnom, double t)
{
double kl = 0.1;
double kv = 0.5;
double Fcopr = kv*(V - Vnom) + kl*(Ll - Lnom);
return Fcopr;
}

static double Fyprnom(double LL, double Ln)
{
double Fyp ;
double cl = 1000;
if(LL-Ln>0)
{Fyp = (cl/Ln)*(LL-Ln);}
else
{Fyp = 0;}
return Fyp;
}

};



A = gY; //Первоначальная инициализация вектора правых частей
int n =gY.size();
double mtrl =mtr/(n-2);
double *r = new double[n-1]; //Радиус-вектора концевых точек
double *L = new double[n-2]; //Длины участков

// Определения длины радиус-векторов
for(int i = 0; i<=n-2; i++)
{
r[i] = Forces::ModVect(gY[i][0],gY[i][2], gY[i][4]);
}
// Длина участка троса между точками
for(int i = 0; i<n-2; i++)
{
```

```
L[i] = Forces::ModVect(gY[i+1][0] - gY[i][0], gY[i+1][2] - gY[i][2], gY[i+1][4] - gY[i][4]);
}

//Длина развернувшейся части
Lsum = 0;
for(int i = 0; i<n-2; i++)
{Lsum = Lsum+L[i];}

//Растояние между концевыми телами
double Lkask = Forces::ModVect(gY[n-2][0] - gY[0][0], gY[n-2][2] - gY[0][2], gY[n-2][4] - gY[0][4]);

// Правые части Диф. уравнений
A[0][0] = gY[0][1];
A[0][1] = (-mu/(r[0]*r[0]))*gY[0][0]/r[0] - (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][0] - gY[1][0])/L[0])/mka;
A[0][2] = gY[0][3];

A[0][3] = (-mu/(r[0]*r[0]))*gY[0][2]/r[0] - (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][2] - gY[1][2])/L[0])/mka;
A[0][4] = gY[0][5];
A[0][5] = (-mu/(r[0]*r[0]))*gY[0][4]/r[0] - (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][4] - gY[1][4])/L[0])/mka;

A[1][0] = gY[1][1];
A[1][1] = (-mu/(r[1]*r[1]))*gY[1][0]/r[1] + (Forces:: Fypr(Lsum,gY[n-1][4])*(gY[0][0] - gY[1][0])/L[0] - Forces::Fyprnom(L[1], gLnom)*(gY[1][0] - gY[2][0])/L[1])/(mtrl/2);
A[1][2] = gY[1][3];
A[1][3] = (-mu/(r[1]*r[1]))*gY[1][2]/r[1] + (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][2] - gY[1][2])/L[0] - Forces::Fyprnom(L[1], gLnom)*(gY[1][2] - gY[2][2])/L[1])/(mtrl/2);
A[1][4] = gY[1][5];
A[1][5] = (-mu/(r[1]*r[1]))*gY[1][4]/r[1] + (Forces::Fypr(Lsum,gY[n-1][4])*(gY[0][4] - gY[1][4])/L[0] - Forces::Fyprnom(L[1], gLnom)*(gY[1][4] - gY[2][4])/L[1])/(mtrl/2);

for(int i=2; i<n-2; i++)
{
A[i][0] = gY[i][1];
A[i][1] = (-mu/(r[i]*r[i]))*gY[i][0]/r[i] + (Forces::Fyprnom(L[i-1], gLnom)*(gY[i-1][0] - gY[i][0])/L[i-1] - Forces::Fyprnom(L[i], gLnom)*(gY[i][0] - gY[i+1][0])/L[i])/(mtrl);
A[i][2] = gY[i][3];
A[i][3] = (-mu/(r[i]*r[i]))*gY[i][2]/r[i] + ( Forces::Fyprnom(L[i-1], gLnom)*(gY[i-1][2] - gY[i][2])/L[i-1] - Forces::Fyprnom(L[i], gLnom)*(gY[i][2] - gY[i+1][2])/L[i])/(mtrl);
A[i][4] = gY[i][5];
A[i][5] = (-mu/(r[i]*r[i]))*gY[i][4]/r[i] + ( Forces::Fyprnom(L[i-1],gLnom)*(gY[i-1][4] - gY[i][4])/L[i-1] - Forces::Fyprnom(L[i], gLnom)*(gY[i][4] - gY[i+1][4])/L[i])/(mtrl);
}

A[n-2][0] = gY[n-2][1];
A[n-2][1] = (-mu/(r[n-2]*r[n-2]))*gY[n-2][0]/r[n-2] + (Forces::Fyprnom(L[n-3], gLnom)*(gY[n-3][0] - gY[n-2][0])/L[n-3])/(mck+mtrl/2);
A[n-2][2] = gY[n-2][3];
```

```
A[n-2][3] = (-mu/(r[n-2]*r[n-2]))*gY[n-2][2]/r[n-2] + (Forces::Fyprnom(L[n-3], gLnom)*(gY[n-3][2] - gY[n-2][2])/L[n-3])/(mck+mtrl/2);
A[n-2][4] = gY[n-2][5];
A[n-2][5] = (-mu/(r[n-2]*r[n-2]))*gY[n-2][4]/r[n-2] + (Forces::Fyprnom(L[n-3], gLnom)*(gY[n-3][4] - gY[n-2][4])/L[n-3])/(mck+mtrl/2);

A[n-1][0] = gY[n-1][1];
A[n-1][1] = -2*(gY[n-1][3]/gY[n-1][2])*(PSI+gY[n-1][1]) - 1.5*PSI*PSI*sin(2*gY[n-1][0]);
A[n-1][2] = gY[n-1][3];
A[n-1][3] = gY[n-1][2]*(pow((gY[n-1][1]+PSI),2)-pow(PSI,2)*(1-3*cos(gY[n-1][0])*cos(gY[n-1][0]))) - Forces::Fc(gY[n-1][3],gY[n-1][2],gt)/mck;
A[n-1][4] = gY[n-1][5];
A[n-1][5] = (rkat/Ikat)*(-Forces::Fccc( gY[n-1][5],gY[n-1][3],gY[n-1][4],gY[n-1][2], gt) + Forces::Fypr(Lsum,gY[n-1][4]));

delete []r;
delete []L;
```
