Полет на Луну
Или
Возвращаемся к старому доброму Эйлеру

Очков В.Ф., Петухов В.Н.

«Полёт на Луну» – так назывался советский мультфильм 1953 года, два кадра которого показаны на рис. 1. Его можно посмотреть в интернете – https://yandex.ru/video/preview/5477323758367398669. Фильм наивный, но довольно забавный, если принять во внимание, что в 1957 году был запущен первый искусственный спутник Земли, а в 1961 году Юрий Гагарин полетел в космос…

В те далекие времена почти все мальчишки, включая и одного из авторов этой статьи, мечтали стать космонавтами и так рисовали старт космических аппаратов: ракета разгоняется по длинной параболической эстакаде и по диагонали взмывает ввысь «навстречу звездам» – см. рис. 1.

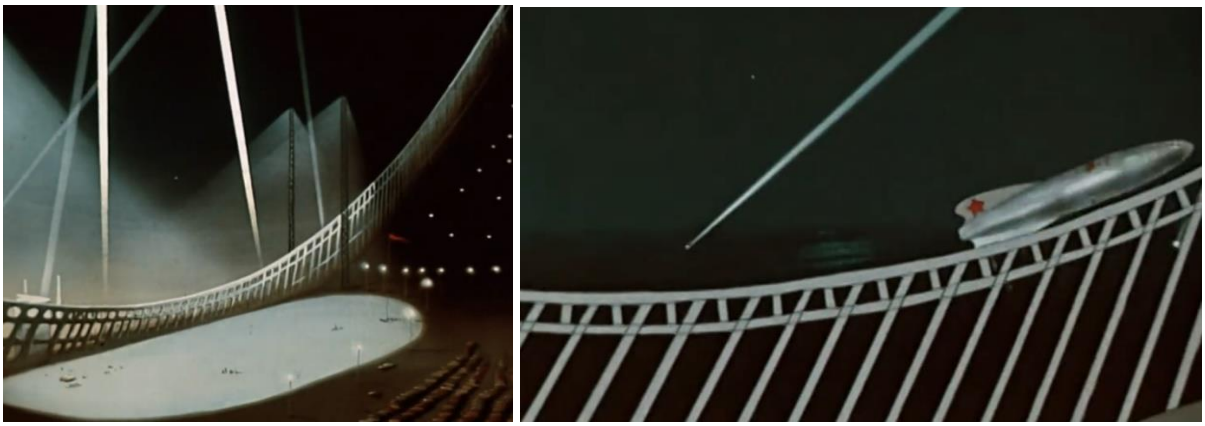


Рис. 1. Два кадра из мультфильма «Полет на Луну»

Но когда это автор, будучи уже не ребёнком, а юношей впервые увидел кадры кинохроники пуска реальной ракеты, то он очень удивился и разочаровался. Никакой романтики! Ракета, привязанная к какому-то столбу, отвязывается от него и как бы нехотя отрывается от поверхности Земли и медленно в клубах дыма поднимаясь вверх (см. рис. 6). Дело в том, что до конца 60-х годов прошлого века в Советском Союзе по соображениям секретности кинохронику пуска ракет не показывали ни в кинотеатрах, ни по телевизору. Поэтому-то ребята и рисовали фантастические огромные эстакады из знаменитого мультфильма. Секретность тех времён мы ещё упомянем.

А давайте рассчитаем пуск ракеты – изменение во времени её высоты и скорости. Заодно покажем, зачем ракеты по совету Циолковского делают многоступенчатыми. И будем использовать для сравнения два компьютерных инструмента – физико-математическую программу SMath Studio (www.smath.com) и среду визуального структурного моделирования SimInTech (www.simintech.ru).

На рисунке 2 показан такой расчёт в среде SMath Studio для одноступенчатой ракеты. Исходные данные (операторы с ярлыками-комментариями) условные. Читатель может ими «поиграть» и посмотреть, что будет получаться.

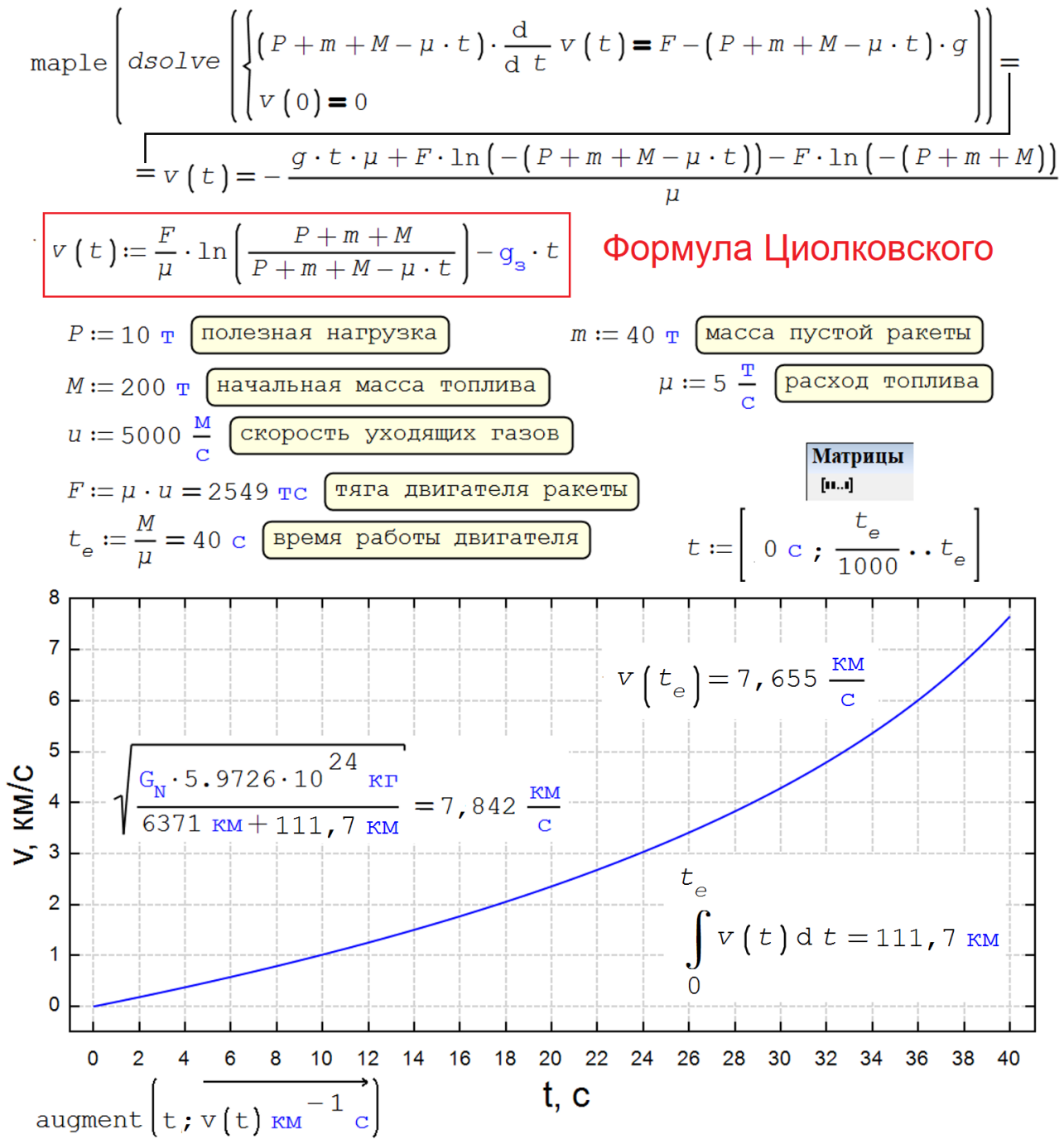


Рис. 2. Расчёт полета одноступенчатой ракеты

На рисунке 2 до ввода исходных данных (числа с единицами измерения) аналитически решается обыкновенное дифференциальное уравнение (частный случай уравнения Мещерского), отображающее второй закон Ньютона. Масса материальной точки (физического объекта, имеющего массу, но не имеющего размера) со временем уменьшается за счет сгорания топлива и окислителя. Эта масса умножается на ускорение – на первую

производную скорости v по времени t (левая часть уравнения). На ракету действуют две силы в противоположных направлениях – тяга ракетного двигателя (F) и сила притяжения Земли (правая часть уравнения). Первая сила – это константа (одно из упрощений задачи), а вторая уменьшается за счёт сгорания топлива и окислителя. Константа g ($g_з$) – это ускорение свободного падения, а выражение в скобках – это уменьшающаяся масса ракеты. Кстати, в советском мультфильме можно видеть, что двигатель фантастической ракеты импульсный. У нас же ракета будет с двигателем непрерывного действия.

В среде SMath нет средств аналитического решения дифференциальных уравнений. Они есть в пакете Maple, который как плагин подключается к SMath через команду меню Сервис/Дополнения/Галерея онлайн. Выданное функцией `dsolve` решение вручную дорабатывается (разность логарифмов равна логарифму дроби) и формирует функцию пользователя $v(t)$, которая отображается на графике. Мы получили знаменитую формулу Циолковского с натуральным логарифмом отношения двух масс ракеты для двух моментов времени.

Внутри графика на рисунке 2 помещены тройка расчётных операторов, из которых видно, что наша ракета поднимется на высоту почти 112 км, но не достигнет первой космической скорости 7,842 км/с и не выведет на орбиту искусственный спутник Земли. Формула первой космической скорости взята из интернета, но её при желании тоже можно вывести. А для вычисления пройденного пути используется определенный интеграл.

А как всё-таки вывести спутник на орбиту Земли!? Можно увеличивать запас топлива и окислителя, а можно поступить иначе – запустить в ближний космос двухступенчатую ракету, использующую такое же или даже меньшее количество топлива и окислителя.

Расчёт, показанный на рис. 3, отличается от предыдущего тем, что некоторые исходные данные представлены не в виде скаляров, а в виде векторов с двумя элементами, хранящими параметры двух ступеней ракеты. Кроме того, введены две функции-ступеньки, возвращающие массу ракеты $m(t)$ и её тягу $F(t)$ в зависимости от времени. В эти функции введен оператор `if`, который затруднит аналитическое решение задачи по шаблону, показанному на рис. 2. Можно, конечно, разбить задачу на две части и найти отдельно формулы Циолковского для стартовых двух ступеней и для второй ступени ракеты. Но если в дальнейшем учитывать сопротивление воздуха, изменение по высоте ускорения свободного

падения и другие нюансы, то аналитика станет бессильной. Нашу задачу нужно будет решать числено, а не аналитически. А каким методом?

В настоящее время наблюдается некий вычислительный ренессанс[1]. Пользователи современных быстродействующих компьютеров возвращаются к простым и понятным алгоритмам. Раньше они применялись редко в том числе и из-за медленного счёта старых компьютеров. В настоящее время это ограничение существенно ослаблено в первую очередь для задач учебного плана, где недоговорённость нежелательна.

Вернемся в славную эпоху запуска первых космических аппаратов. В те времена происходило не только «покорение космоса», но и наблюдался революционный переход от ручных расчётов к расчётам на электронных вычислительных машинах – цифровых и аналоговых. Если говорить о цифровых электронных вычислительных машинах (ЭЦВМ – иностранное слово «компьютер» тогда избегали употреблять), то они работали по программам, использующим различные алгоритмы, в том числе и по тем, какие были предложены несколько веков назад для ручных вычислений. Началась гонка не только в космосе, но и на Земле – создавались все более точные и быстрые программы численного решения тех же дифференциальных уравнений, описывающих полёт космических аппаратов. Самый первый и самый простой метод численного решения обыкновенного дифференциального уравнения в 1768 году предложил великий Эйлер (1707–1783). В то время он работал в России. Но этот метод как-то ушел в тень. Его оттеснили всякие Рунге, Кутты, Адамсы, Левенберги, Марквардты… К сожалению, фамилий советских ученых в этом списке нет. Отчасти повторилась старая история с уравнением Мещерского (см. ниже), усугубившаяся условиями секретности[2].

У нас компьютер быстрый, да и задача довольно простая – расчёт полёта двухступенчатой ракеты. Давайте решим её методом Эйлера!

Его суть предельно проста[3], и в этом его главное преимущество. Нам известно значение искомой функции (скорости полета ракеты) в предыдущей точке и значение производной

---

[1] Очков В.Ф., Чудова Ю.В. Вычислительный ренесанс: метод Ньютона // Энергия: экономика, техника, экология. № 9. 2023. С. 7-16 (http://www.twt.mpei.ac.ru/ochkov/EEE-9-2023-Newton.pdf)

[2] На закате Советской власти выдавали талоны за сданную макулатуру для приобретение дефицитных книг, многие из которых, кстати, сами были макулатурой. Говорят, что в эти застойные времена в утиль отнесли много пакетов перфокарт и мотков перфолент с ценнейшими программами для ЭВМ.

[3] https://ru.wikipedia.org/wiki/Метод_Эйлера

искомой функции в этой точке, и мы можем через производную оценить значение этой функции в последующей точке. Этот метод можно было бы назвать методом касательных (вернее, методом касательной), если бы это название не было бы вторым названием метода Ньютона (см. выше), предназначенного для численного решения алгебраических, а не дифференциальных уравнений.

В настоящее время скорость даже такого компьютера как смартфон[4], несравнимо выше скорости первых компьютеров. Это и определяет отмеченный ренессанс в вычислениях. Усложненные быстрые алгоритмы и программы имеют один существенный недостаток – их трудно понять непосвященному. Это вносит некий дискомфорт в процесс решения задачи. А тут должна быть радость и удовольствие.

На рисунке 3 сразу за функциями `m(t)` и `F(t)` введена функция `a(t)` – производная искомой функции `v` (a: acceleraton – ускорение). Далее записана небольшая программа с циклом `for`, генерирующая матрицу `tv` (время и скорость), из которой затем изымаются векторы `t` и `v` для построения графика изменения скорости ракеты по высоте.

Конечная скорость двухступенчатой ракеты (10,91 км/с) с запасом превысила первую космическую. Ура! Спутник вышел на орбиту, а ракета использовала те же 200 тонн топлива и окислителя.

[4] Программа SMath может работать и на смартфоне.

$P := 10\ t \qquad m := [\,40\ \ 5\,]\ t \qquad M := [\,160\ \ 40\,]\ t \qquad \mu := [\,5\ \ 2\,]\ \frac{t}{s} \qquad u := 5000\ \frac{m}{s}$

$\Delta t := \frac{\overrightarrow{M}}{\mu} = [\,32\ s\ \ 20\ s\,] \qquad t_e := \sum \Delta t = 52\ s$ sum sum (1)

$$m(t) := P + \text{if } t < \Delta t_1,\ \left(\sum (M + m)\right) - \mu_1 \cdot t,\ \text{else},\ M_2 + m_2 - \mu_2 \cdot \left(t - \Delta t_1\right)$$

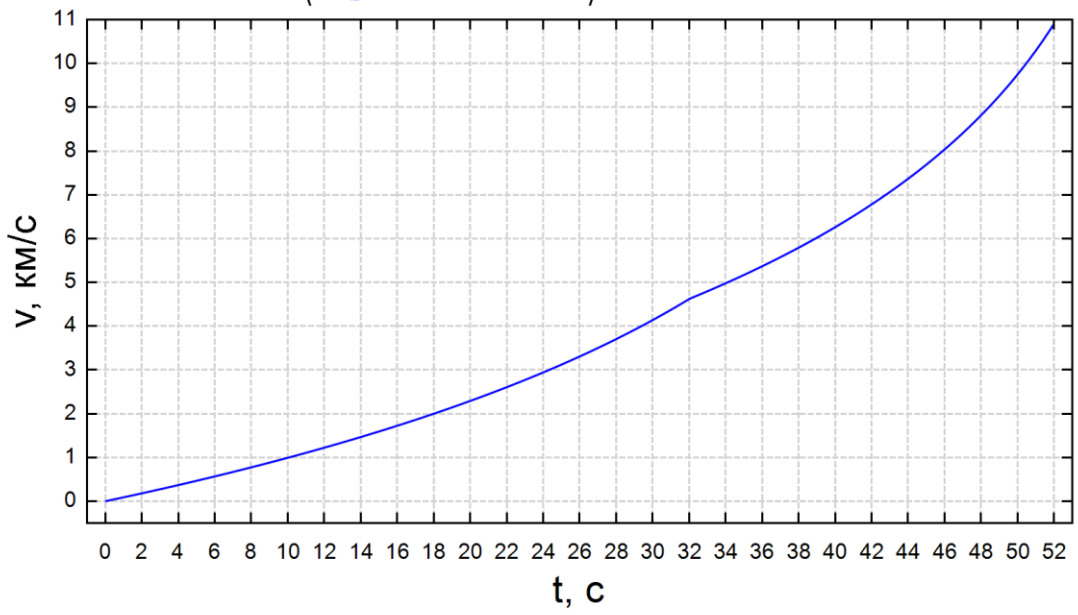
$$F(t) := u \cdot \text{if } t < \Delta t_1,\ \mu_1,\ \text{else},\ \mu_2$$

масса на ускарение равно сумме приложенных сил

$m(t) \cdot a(t) = F(t) - m(t)\ g_e \Rightarrow a(t) := \frac{F(t) - m(t)\ g_e}{m(t)}$

$$Euler(y_b, x_b, x_e, n, a) := \left| \begin{array}{l} \left[X_1 := x_b \quad Y_1 := y_b \quad \Delta := \frac{x_e - x_b}{n} \right] \\ \text{for } i \in [2..(n+1)] \\ \quad \left| \begin{array}{l} Y_i := Y_{i-1} + a\left(X_{i-1}\right) \cdot \Delta \\ X_i := X_{i-1} + \Delta \end{array} \right. \\ \text{augment}(X, Y) \end{array} \right.$$

$n := 1000 \qquad tv := Euler\left(0\ \frac{m}{s},\ 0\ s,\ t_e,\ n,\ a\right) \qquad t := \text{col}(tv, 1) \qquad v := \text{col}(tv, 2)$



$\text{augment}\left(\frac{t}{s},\ \frac{v}{km\ s^{-1}}\right) \qquad \max(v) = 10.91\ \frac{km}{s}$

$v(\tau) := \text{lspline}(t, v, \tau)\ \frac{m}{s} \qquad \int_0^{t_e} v(\tau)\, d\tau = 202.8\ km$

Рис. 3. Расчёт полета двухступенчатой ракеты

На рисунке 4 дана визуальная оценка точности метода Эйлера: синяя гладкая кривая – это аналитическое решений задачи о полёте одноступенчатой ракеты, а красная ломанная линия с точками – это результат численного решения задачи при n = 20 (**студентам необходимо самостоятельно создать аргумент этого графика**). Если же значение n увеличить хотя бы до 100, то эти две кривые практически сольются в экстазе, пардон, в одну линию. А на рис. 3 значение n было дано с большим избытком – 1000! И на время счёта это практически не повлияло.

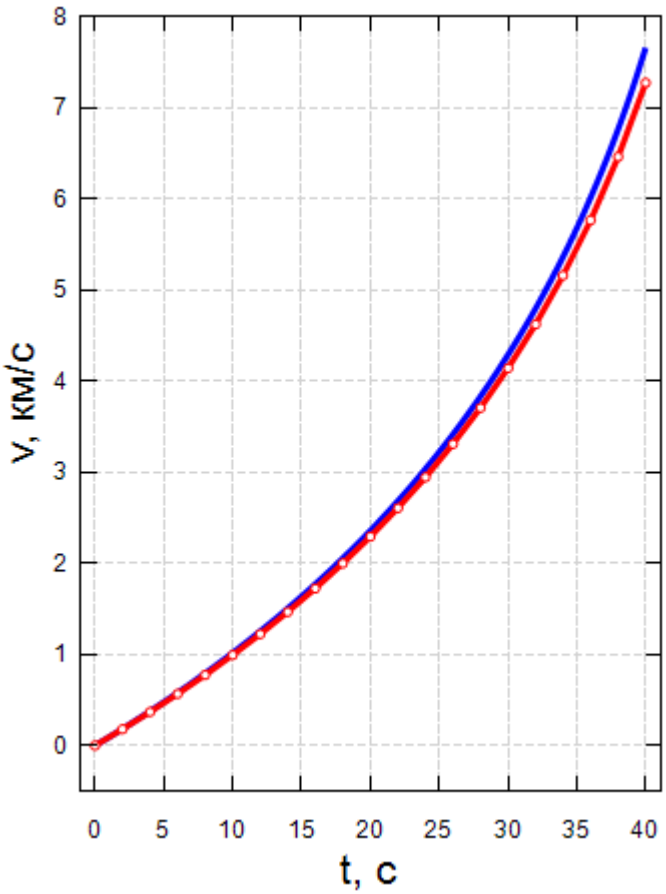


Рис. 4. Визуальная оценка точности метода Эйлера: синяя кривая – точное аналитическое решение, красная кривая с точками – численный метод Эйлера

Но если б во времена первых полётов в космос расчётчик пришел бы в вычислительный центр своего НИИ, принес пакет перфокарт (см. сноску 2) и сказал, что ему нужно примерно десять часов машинного времени, чтобы подсчитать задачу, подобную вышеописанной, то его бы выгнали из машинного зала и при этом улюлюкали бы и топали ногами. Расчетчика

заставили бы придумать новую программу, которая считала бы задачу не за десять часов, а хотя бы за два-три часа. Настолько было дорого машинное время в те времена. Это сейчас мы порой не знаем, чем занять наш персональный компьютер, а в те легендарные времена такой проблемы не было, почти не было. Это одна из причин появления программ, в которых сейчас мало кто разберется. Отсюда и вышеописанный программный ренессанс!

А вот как задача о полете ракеты может быть решена в среде SimInThech.

Решим задачу с одноступенчатой ракетой двумя способами.

1) Сначала используем формулу Циолковского (см. рис. 2), которая позволяет рассчитать скорость ракеты в любой момент времени:

$$v(t) = \frac{F}{\mu} \cdot \ln\left(\frac{P + m + M}{P + m + M - \mu \cdot t}\right) - g \cdot t$$

Где:
P = 10 000 кг – полезная нагрузка;
M = 200 000 кг – начальная масса топлива;
u = 5 000 м/с – скорость уходящих газов;
μ = 5 000 кг/с – расход топлива;
$F = u \cdot \mu$ – тяга двигателя ракеты.

Будем использовать блок «Язык программирования (PL)» (рис. 5). Он выполняет вычисления, используя данные в линиях связи на входе, и выкладывает результат в линии связи на выходе.

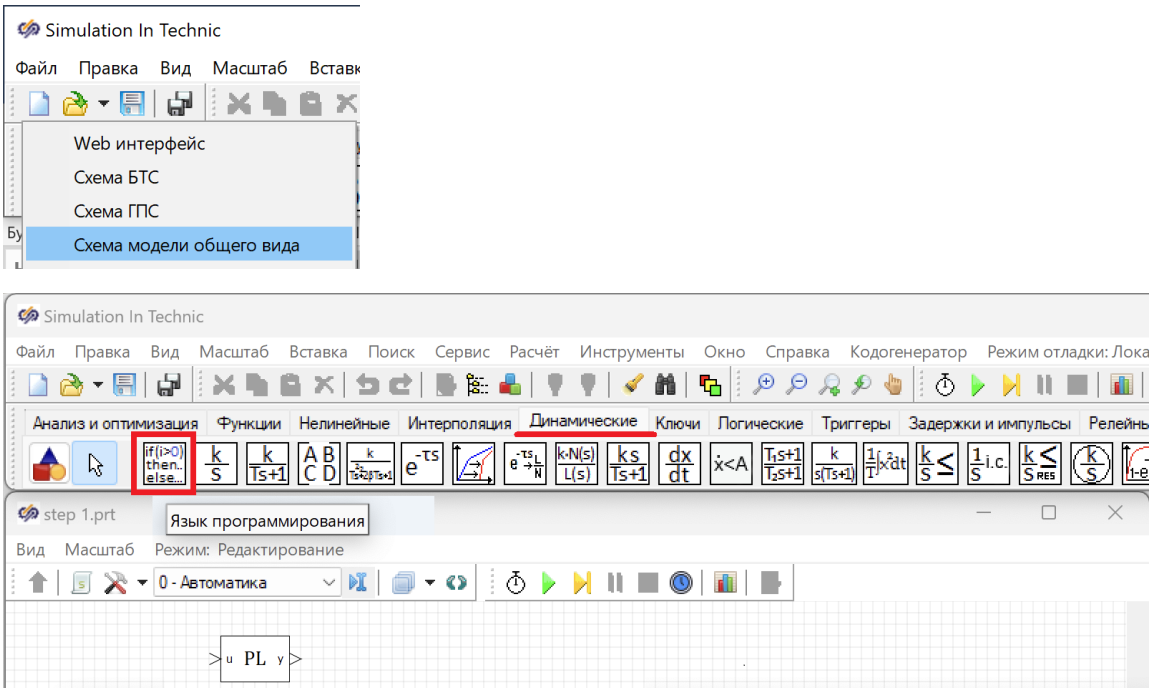


Рис. 5. Вставка в проект блока «Язык программирования (PL)»

Если кликнут по блоку два раза, то вместо свойства (как для других блоков) появляется окно редактирования, где мы можем написать программу на языке SimInTech – см. рис. 6.

Заменим текст по умолчанию в блоке «Язык программирования». Запишем параметры ракеты в список констант, разделенных запятыми, а дальше запишем формулу Циолковского. На выход отправим переменную скорость v.

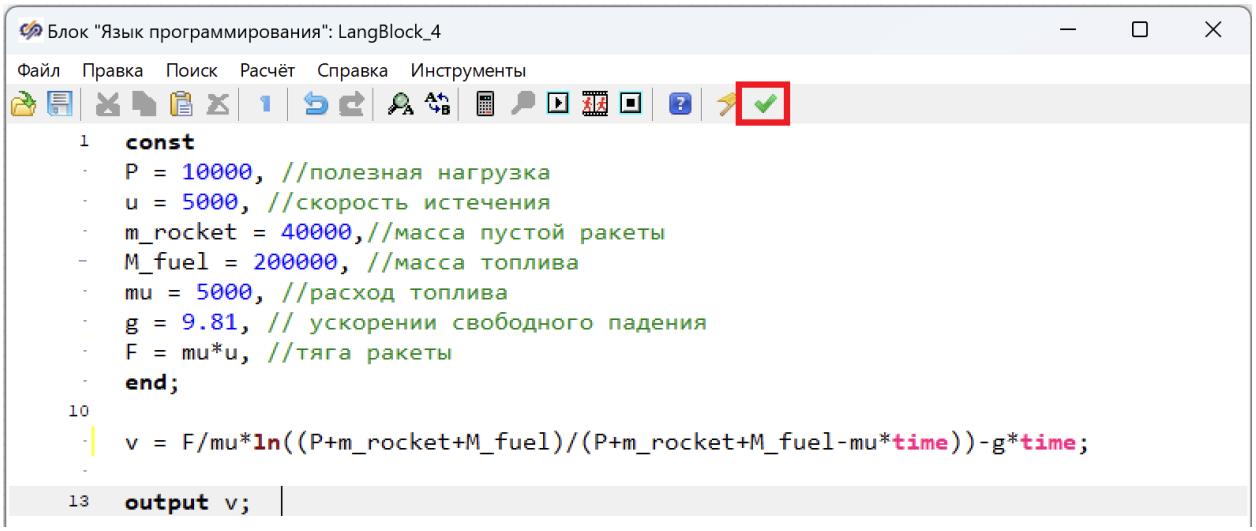


Рис. 6.

Во время расчета этот блок будет выдавать значение скорости с учетом времени. Глобальная переменная **time** на каждом шаге содержит текущее модельное время.

Переменные, которые мы хотим выдать из блока, перечисляются после ключевого слова **output**. Если сохранить модель нажав на зеленую галочку верху, то в блоке (рис. 5) исчезнет вход, а выход будет подписан как **v**.

Подключим к выходу блока PL блок интегратор ($\frac{k}{s}$ – рис .7). Если на входе в интегратор, будет скорость, то на выходе будет путь, в нашем случае высота. Выходы и входы при этом нужно соединить линиями, как показано на рис. 7

Выведем на график скорость и высоту для этого подключим график, как показано на рисунке7.

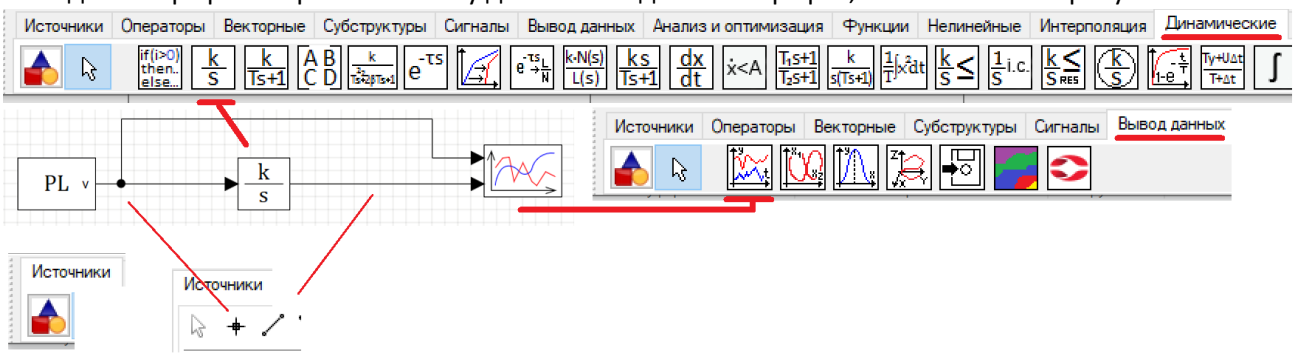
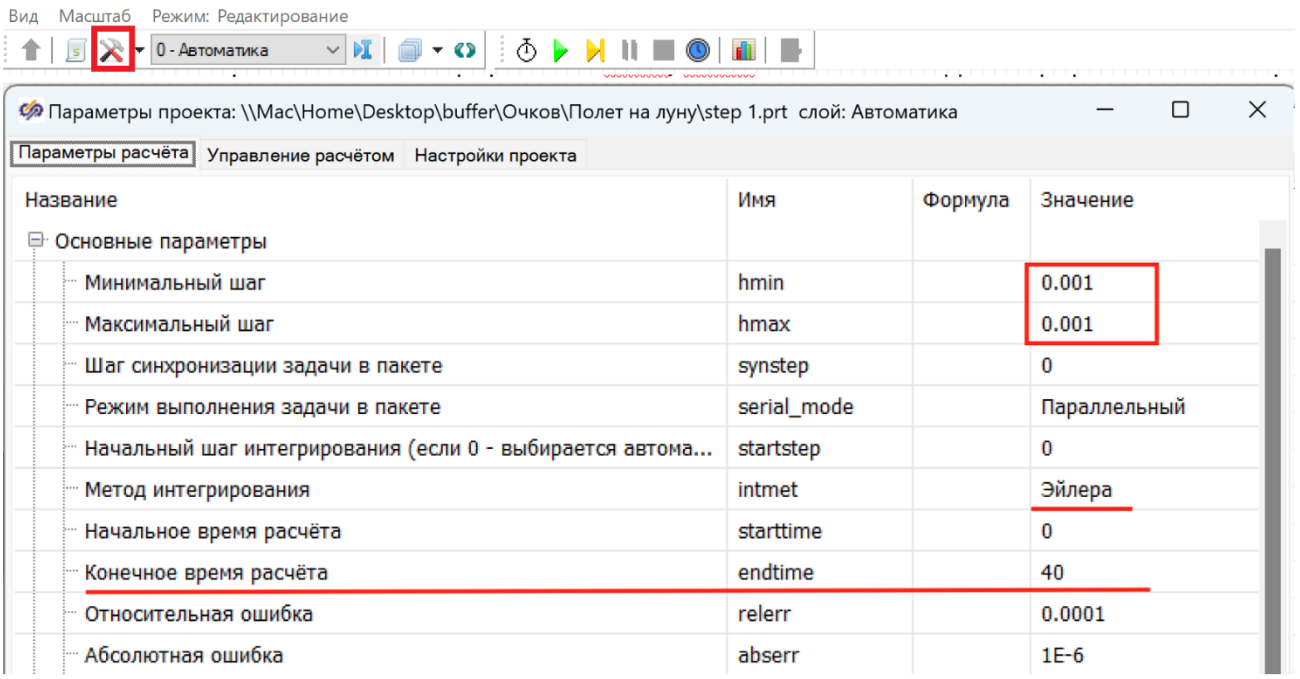


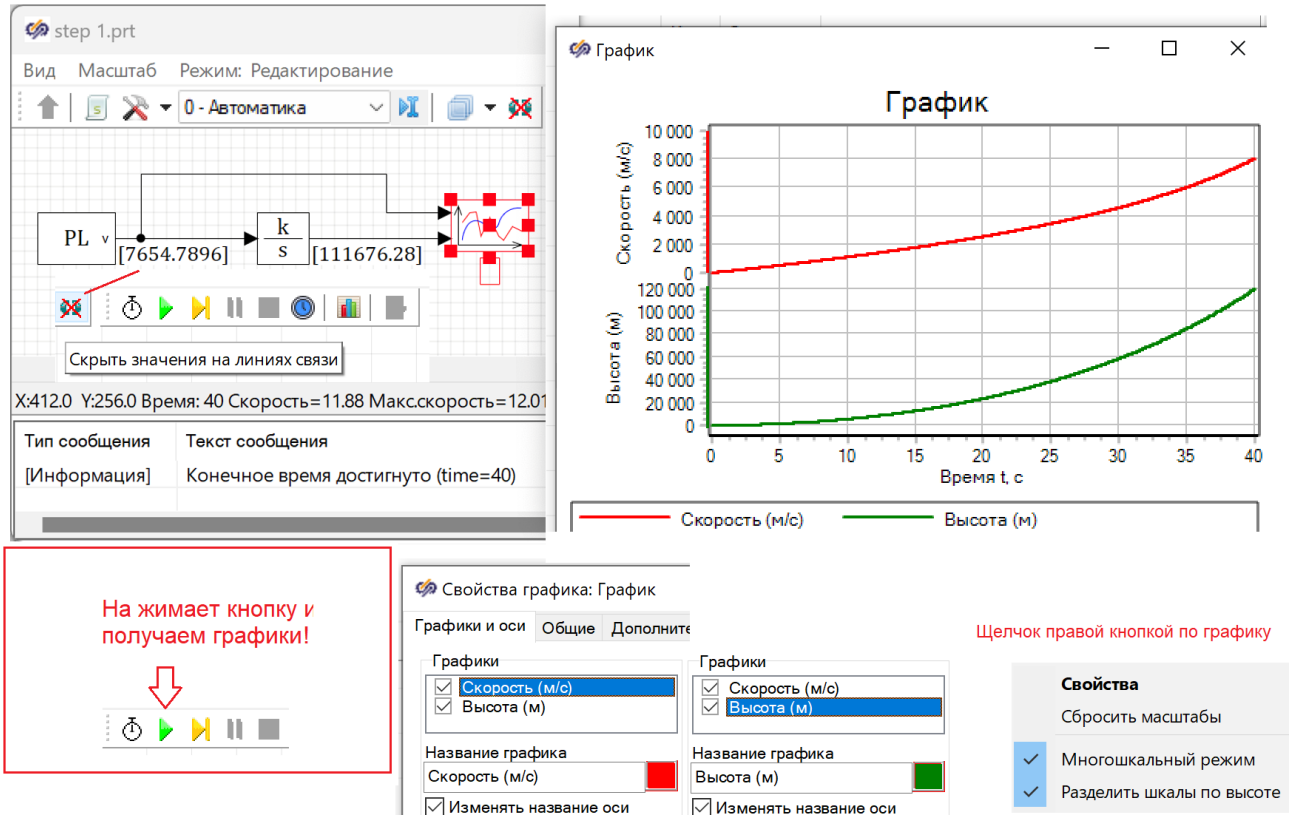
Рис. 7

Перейдем к настройке параметров расчёта и установим конечное время расчета – 40 секунд .

Обратим внимание на настройки шага интегрирования **hmin, hmax = 0.001** и метод интегрирования – Эйлера.



Запускаем расчет и видим, что у нас получилась скорость в конце расчета:
скорость – 7.655 км/с; высота – 111,7 км



Решим интегрировать или не интегрировать вот в чем вопрос?

Но мы не зря назвали наш блок волшебным, он может не только выдавать прямую формулу в зависимости от времени, но может и сам считать интеграл.

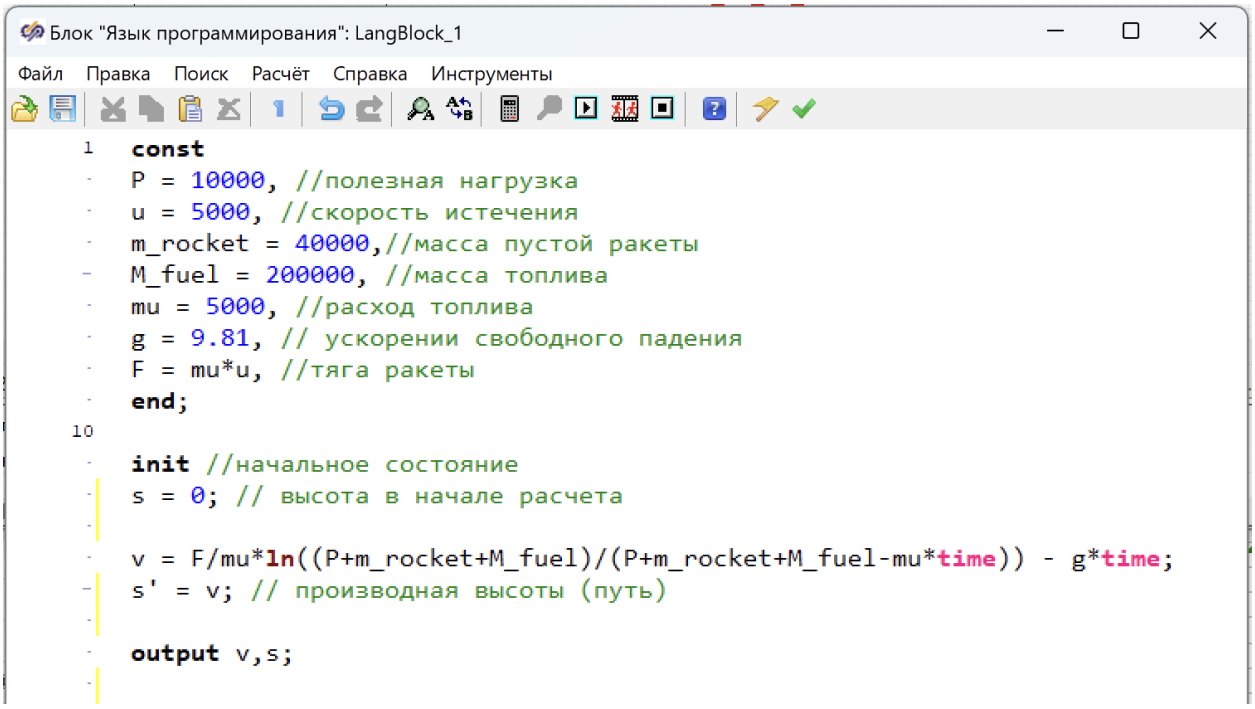
Скопируем блок и поставим на схему копию блока, которую будем модифицировать.

Нам достаточно внести следующие изменения в программу на языке программирования:

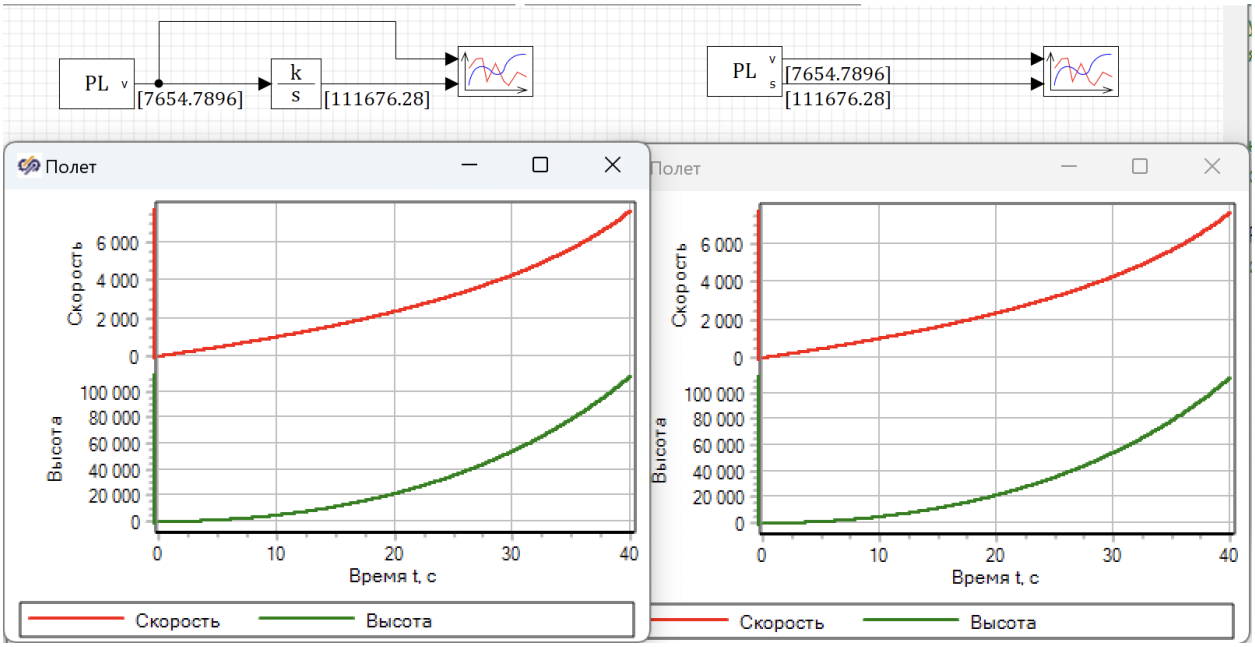
Поскольку скорость у нас — это производная от пути то, мы можем так и записать выражение для производной пути **s’ = v**.

Так же необходимо указать начальное состояние **s = 0** после специального слова **init**

После этого мы можем указать переменную **s** в выходных переменных, полный текст будет выглядит как на рисунке:



Теперь на из блока PL у нас выходят две линии связи по одной идет скорость, по второй высота, сравним результаты расчета, как не удивительно они полностью совпадают!



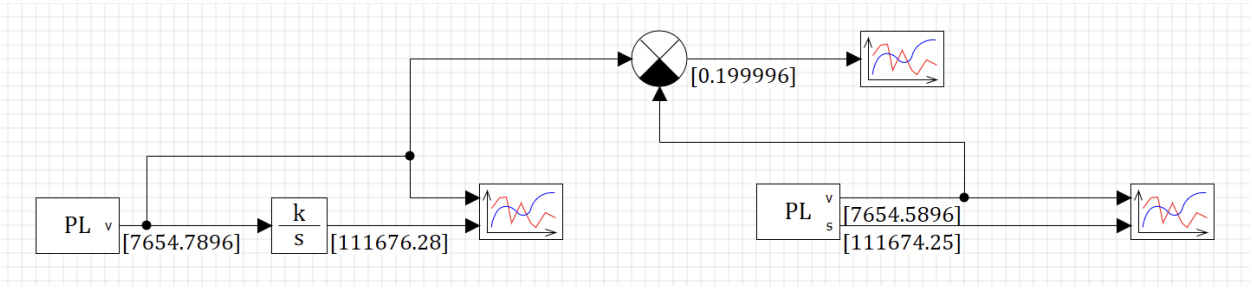
Считаем уравнения физики

Если мы, по скорости в блоке посчитали путь, то что нам мешает посчитать и маму скорость если мы знаем ускорение?
А ускорение мы получим из закона Ньютона:
$F_\Sigma = m \cdot a \rightarrow a = \frac{F_\Sigma}{m}$; где F_Σ – суммарная сила действующая на ракету
Проверим формулу Циолковского, заменив в наше блоке формулу, прямым расчетом физических законов. На ракету у нас действуют всего две силы: отрицательная сила тяжести $-m \cdot g$ и сила тяги F
$F_Z = F - m \cdot g$
Используем эту формулу что бы рассчитать производную и скорость в языке программирования.
Поменяем текст в блоке, где мы рассчитывали пути через производную
Не забудем, что в формуле у нас так же постоянно меняется и масса ракеты, поэтому для нее тоже нужно указать производную, которая равна расходу топлива.

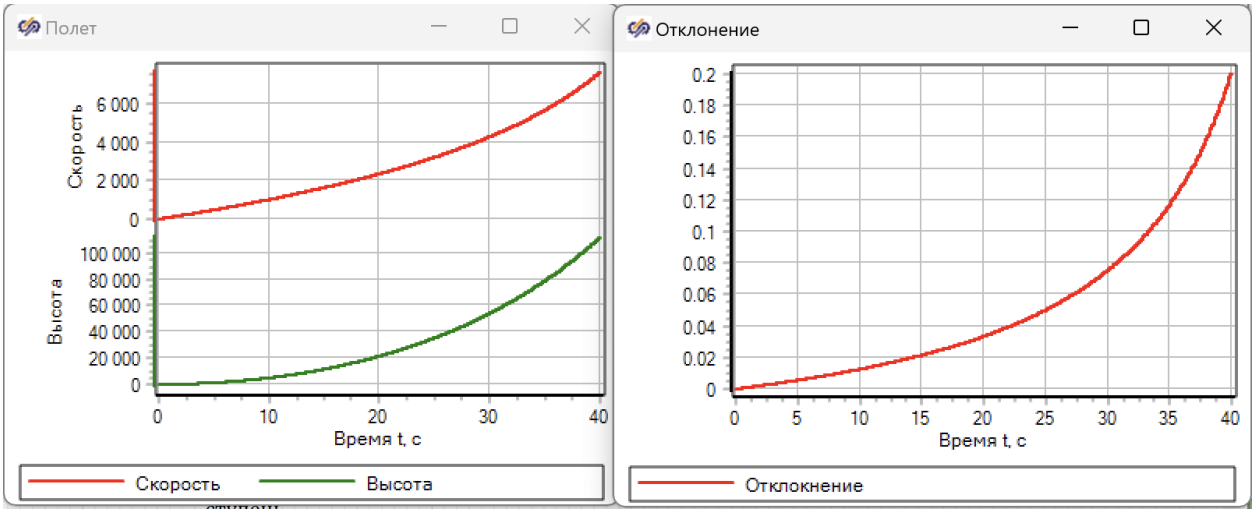
Список констант, описывающих ракету, не поменяется, а вот текст программы будет выглядеть как показной на рисунке:

```
10
   init //начальное состояние
   s = 0, // высота в начале расчета
   v = 0, // скорость на старте
   M = P+m_rocket+M_fuel;  // масса на старте;

   F=mu*u; //сила тяги
   a = (F-g*M)/M; //ускорение
   v' = a;  // производная скорости
   s' = v; // производная высоты
20 M'= -mu; //производная массы

   output v,s;
```

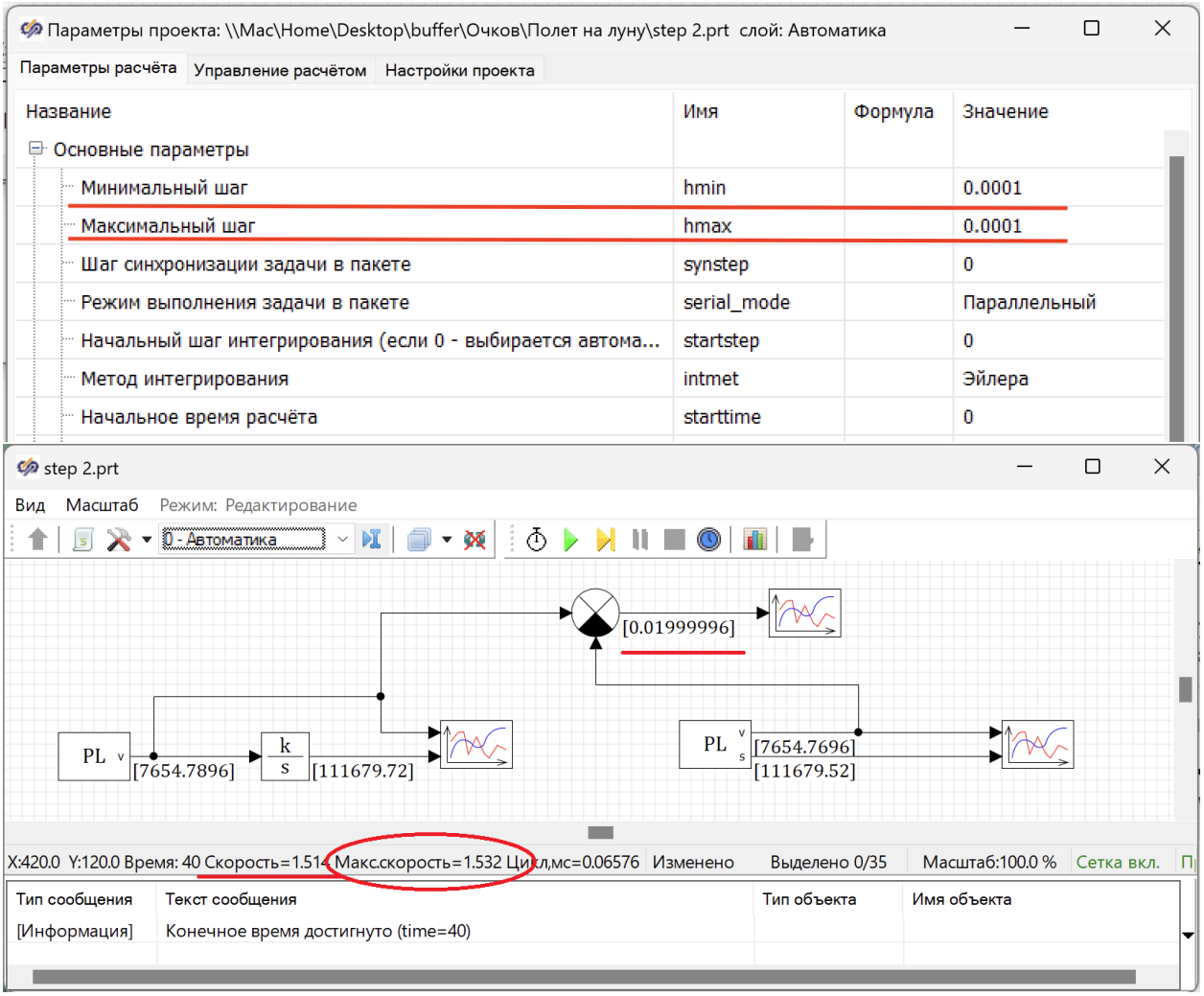
Для того, чтобы сравнит результаты расчета по формуле Циолковского, добавим на схему сравнивающий блок и выведем результат на график:



Запустим на расчет и если, вы все сделали правильно, то мы увидим, что у нас результат расчета отличается. Скорость по формуле Циолковскому - 7 654.7896 м/с, скорость по уравнениям - 7 654.5896 м/с погрешность 0.2 м/с.



Кто тут прав? Не будем скрывать Циолковский прав, и не только потому, что он гений, но и потому, что мы использовали метод Эйлера, описание которого подробно приведено в предыдущей статье. Изобретенный 250 лет назад, он по умолчанию содержит в себе ошибки. Для сокращения ошибок можно уменьшить шаг интегрирования, например зададим минимальный в 10 раз меньше.

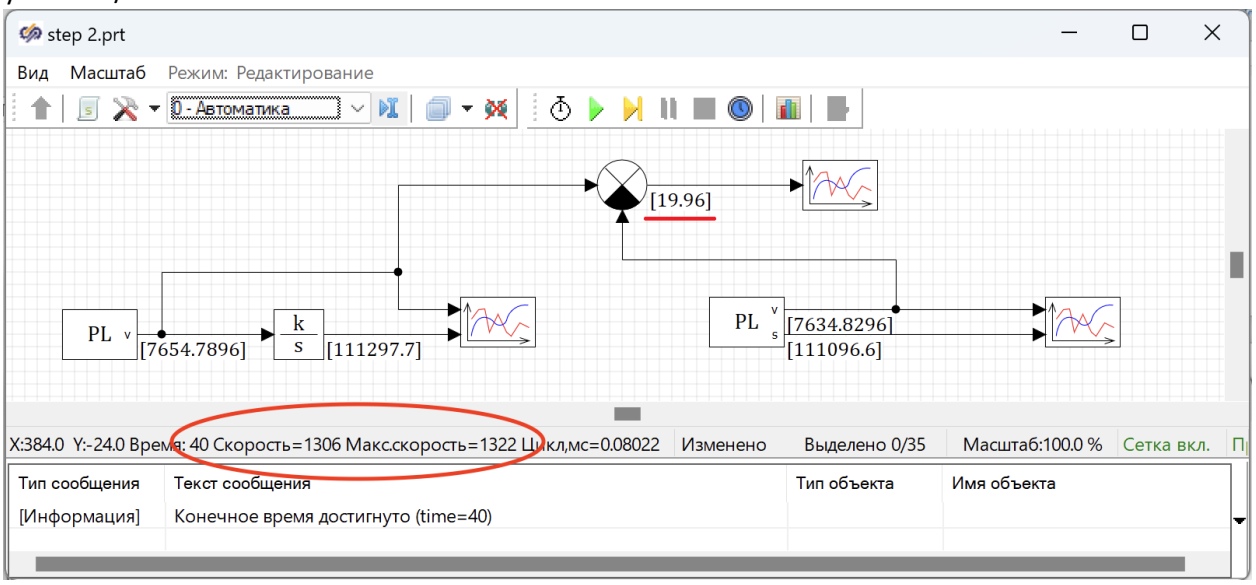


Результат отклонение так же уменьшилось в 10 раз было 0.2 стало 0.02.

Но даже на моем МакBook скорость вычисления значительно замедлилось.

Мы стали считать только в 1.5 быстрее реального времени, а это очень медленно. Скорость можно посмотреть в нижней части схемного окна.

Это кажется логичным что чем больше шагов, по времени, тем больше вычислений требует методы Эйлера. Можно сократить вычисления уменьшить шаг, тогда у нас будет меньше вычислений, например если задать шаг интегрирования 0.1. То вычисление будет мгновенным, нажми на кнопку – получишь результат. Скорость расчета в 1300 раз быстрее реального времени. Но погрешность расчета уже 20 м/с



Получается, как в анекдоте:

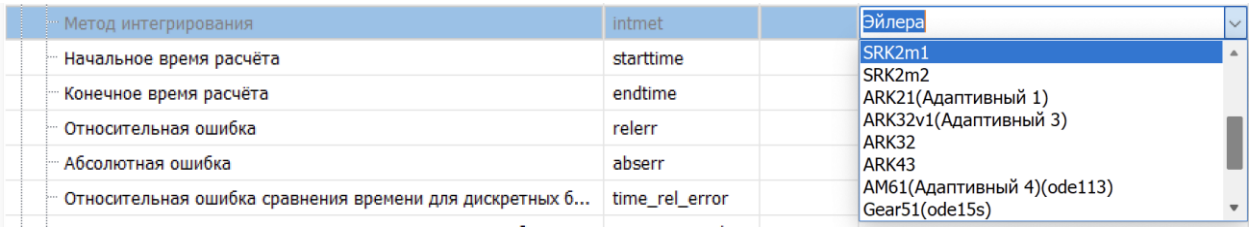
- С какой скоростью вы можете печать
- 20 слов в минуту.
- А можете 50 слов в минуту?
- Могу, и 100 слов минуту, но такая фигня получается.

Казалось бы, некуда деваться, либо быстро, либо точно, либо одно из двух. Но на наше счастье, еще Ломоносов отмечал

«…что может собственных Платонов
И быстрых разумом Невтонов
Российская земля рождать.”

Поэтому кроме не Эйлером единым, вклад Советских и Российских ученых в методы расчет динамических систем ограничивается.

Представляю вам Леонида Марковича Скворцова. Его методы используются в SimInTech и если раскрыть свойства Методы интегрирования, то мы увидим целую пачку методов, с непонятными авиатурами SRK2m1 или ARK32, так вот все эти методы разработаны именно Леонидом Марковичем.



И нельзя сказать, что он секретный физик, специалисты в области численного решения дифференциальных уравнений его не плохо знают. Вот, например статья в цифровой библиотеке, на

сайте Гарварда от 2022 года. «Диагонально-неявные методы Рунге-Кутты третьего и четвертого порядка для жестких и дифференциально-алгебраических задач.»

https://ui.adsabs.harvard.edu/abs/2022CMMPh..62..766S/abstract

А вот свежие статьи уже зарубежных авторов с цитатами от из этой статьи

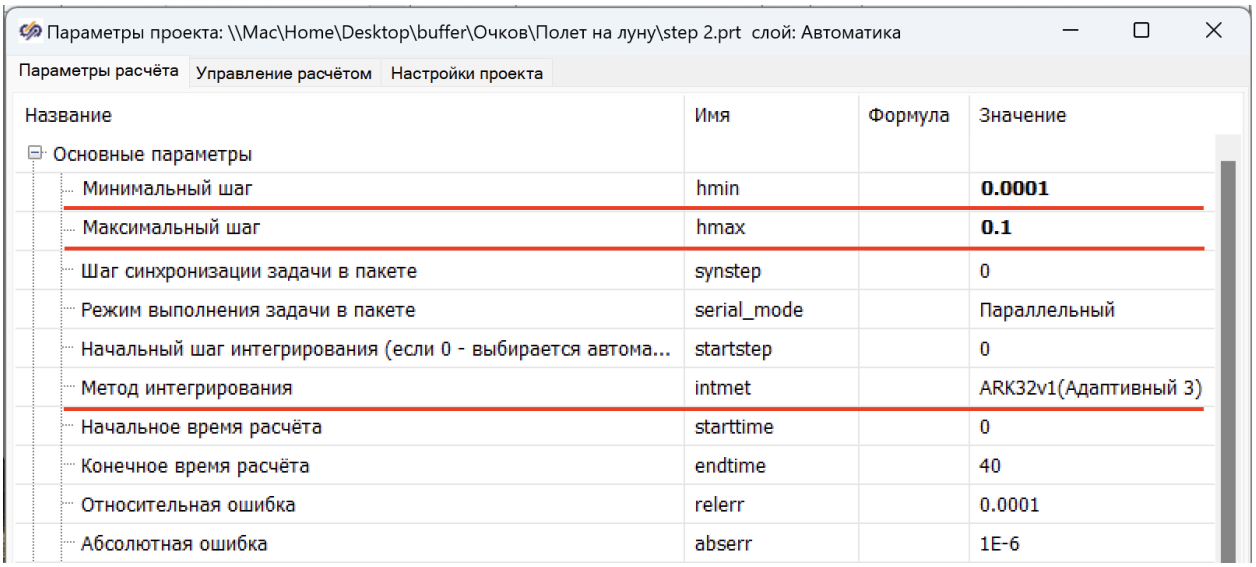
https://ui.adsabs.harvard.edu/abs/2022CMMPh..62..766S/citations

Для чего нужны эти методы? Они обеспечивают возможность менять шаг интегрирования в процессе расчета с тем, что бы можно было ускорить

И пусть вас не вводит в заблуждение имя Рунге-Кутта в название, так называют целый класс методов, а вот конкретная реализации в этом класса — это метод Скворцова.

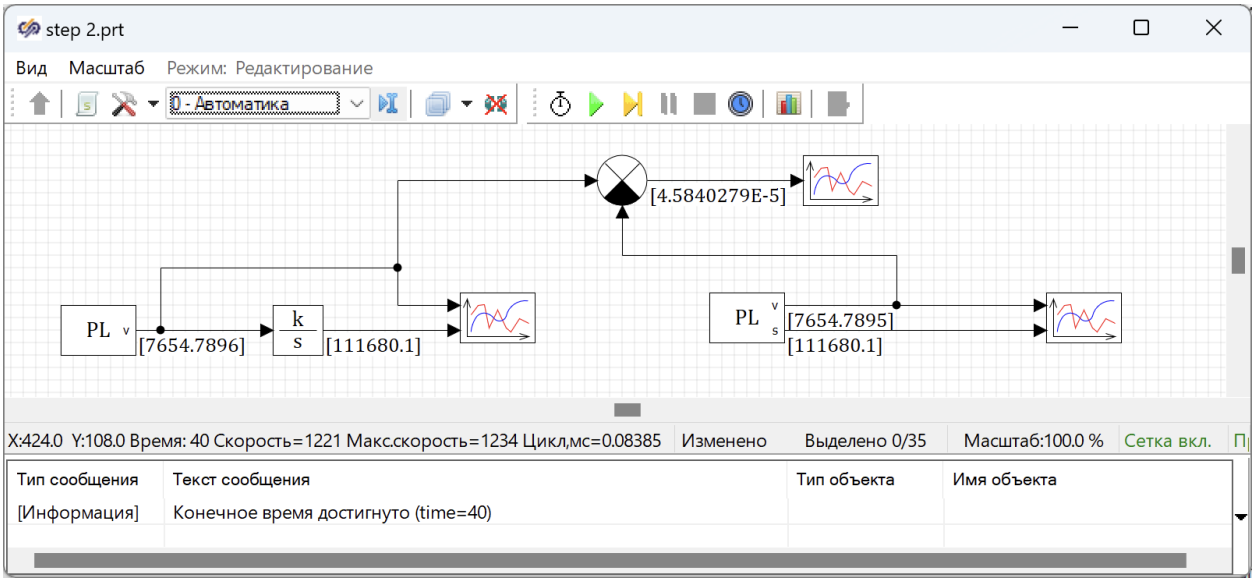
Зачем они нужны? Эти методы позволяют подбирать шаг игнорирования, так что бы увеличить скорость расчета, но не терять точность. Давайте выберем один из методов Скворцова, например (Адаптивный 3).

Зададим минимальный шаг – 0.0001 и максимальны шаг – 0.1



Такой настройкой мы позволим выбирать шаг интегрирования самому методу с тем, чтобы он обеспечила нам заданную точность.

Запустим на расчет и восхитимся результатами получены с помощью советских и российских математиках.



Скорость расчета в 1200 раз быстрее реального времени, а ошибка - 4.6×10^{-5} Именно поэтому методы Скворцова используются в SimInTech, результат как говорится на табло!

Получив полное отличное совпадение с формулой Циолковского, давайте еще более уточним. Вернее, уточним не саму формулу, нашу модель, дело в том, что модель не учитывает сопротивление воздуха. А на больших скоростях, сопротивление воздуха должно вносить коррекцию.

Воспользуемся той же формулой, которую мы использовали в статье для снаряда: $\vec{F} = k \cdot v^2$

Где: v - скорость тела. k - коэффициент сопротивления воздуха.

$$k = \frac{1}{2} \cdot c \cdot \rho \cdot S$$

Здесь: S – площадь лобового сопротивления снаряда;

ρ – плотность воздуха;

Плотность воздуха в зависимости от высоты может рассчитываться специальным блоком в SimInTech.

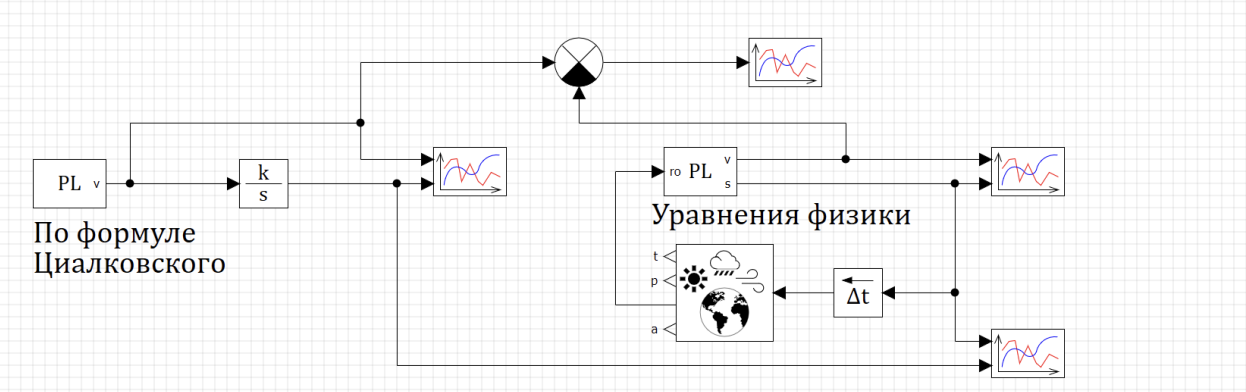
Для того что бы забрать плотность в модель ракеты мы должны прописать служебное слово **input** и дать имя переменной, которую мы будем забирать из блока расчета атмосферы. После этого достаточно добавить недостающие параметры для расчета силы терпения воздуха и добавить силу в уравнения:

Блок "Язык программирования": LangBlock_1

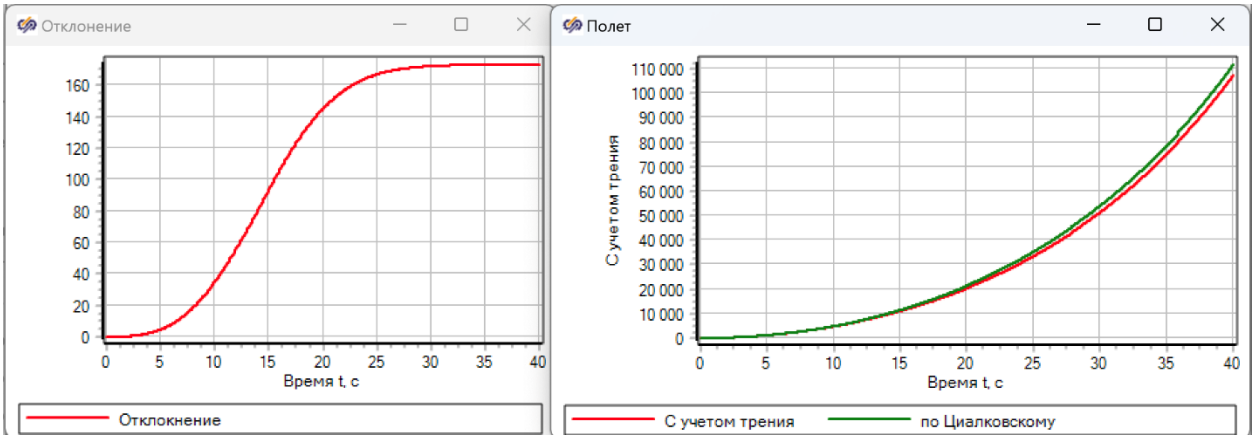
Файл Правка Поиск Расчёт Справка Инструменты

```
input ro;

const
P = 10000, //полезная нагрузка
u = 5000, //скорость истечения
m_rocket = 40000,//масса пустой ракеты
M_fuel = 200000, //масса топлива
mu = 5000, //расход топлива
g = 9.81, // ускорении свободного падения
F = mu*u, //тяга ракеты
c = 0.41, //коэффициент сопротивления
d = 4, // диаметр ракеты
Sr = pi*d^2/4, //площадь лобового сопротивления
end;

init //начальное состояние
s = 0, // высота в начале расчета
v = 0, // скорость на старте
M = P+m_rocket+M_fuel;  // масса на старте;

F_tr = ro*c*Sr*v^2/2; //сила трения
F=mu*u; //сила тяги
a = (F-g*M-F_tr)/M; //ускорение
v' = a;  // производная скорости
s' = v; // производная высоты
M'= -mu; //производная массы

output v,s;
```

Для корректно работы схемы, требуется так же задержка на шаг интегрирования перед блоком расчета свойств атмосферы. Поскольку для расчета высоты, нужна плотность на данной высоте, а для расчета плотности нужна высота, это называется алгебраическая петля, задержка на шаг позволяет исправить это противоречие. В ней мы задаем начальную высоту, а далее на каждом следующем шаге, для расчета полётности мы используем значение высоты с предыдущего шага, сохраненное в задержке. В итоге у нас получается вот такая схема:



Запускаем на расчет и получаем следующие результаты:



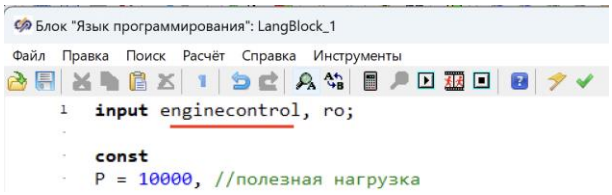
Отклонение, связанное с трение у нас, растем с ростом скорости. Чем больше скорость, тем сильнее трение и тем сильнее отличается скорость по формуле Циолковского от нашего физического расчета. По мере увеличения высоты, плотность падает воздуха падает и примерно на 35 секунде (высота 80 км) трение уже практически не снижает скорость, и отклонение остается постоянный у нас это примерно 172 м/с

По высоте отличие составляет 9 км по идеальной формуле Циолковском высота 111.7 км, по физическому расчете – 107.3 км

Поскольку ракета не вышла на первую космическую скорость, давайте добавим ступени.

Добавим ступени!

Поскольку мы находимся в среде динамического моделирования, то созданную модель множено просто скопировать и получить сразу две ракеты. Конечно, чторбы одна из них была второй ступенью придется немного модифицировать программу в блоке. Во-первых, у нас появляется программа полета. Мы должны иметь возможность включать и выключать двигатель, для этого добавим новую входную переменную **enginecontrol,** добавим ее после служебного слова **input**



Данная переменна будет содержать 1 если двигатель включен и 0 если двигатель выключен.

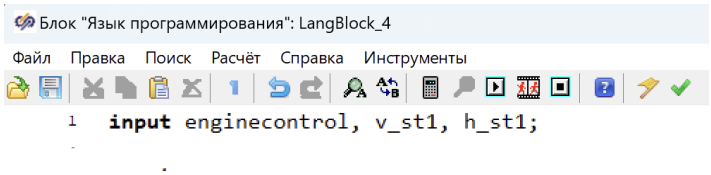
Перейдем к уравнениям физики и учтем режим работы двигателя в расчете сил, действующих на ракету. Добавим новую переменную которая будет менять расход топлива в зависимости от управляющего воздействия, в нашем случае 0 или ! и используем эту переменную в расчёте силы тяги и изменения массы вместо константы.

```
var muc = 5000; //расход топлива контролируемый

init //начальное состояние
s = 0, // высота в начале расчета
v = 0, // скорость на старте
M = P+m_rocket+M_fuel;  // масса на старте;

muc = mu*enginecontrol;
F_tr = sign(v)*ro*c*Sr*v^2/2; //сила трения
F=muc*u; //сила тяги
a = (F-g*M-F_tr)/M; //ускорение
v' = a;  // производная скорости
s' = v; // производная высоты
M'= -muc; //производная массы

output v,s;
```

Теперь у нас расход топлива и соответственно сила тяги, появляются только тогда, когда управляющее воздействие больше 0.

Для второй ступени мы не будем учитывать сопротивление воздуха, поскольку как видно из графика, на высоте больше 80 км, где включается вторая ступень плотность воздуха уже такова, что сопротивление можно не учитывать.

Но нам необходимо учесть высоту, с которой начинает работать вторая ступень, для этого кроме входа управления двигателем мы добавляем два входа, для высоты и скорости, на которой включается двигатель.



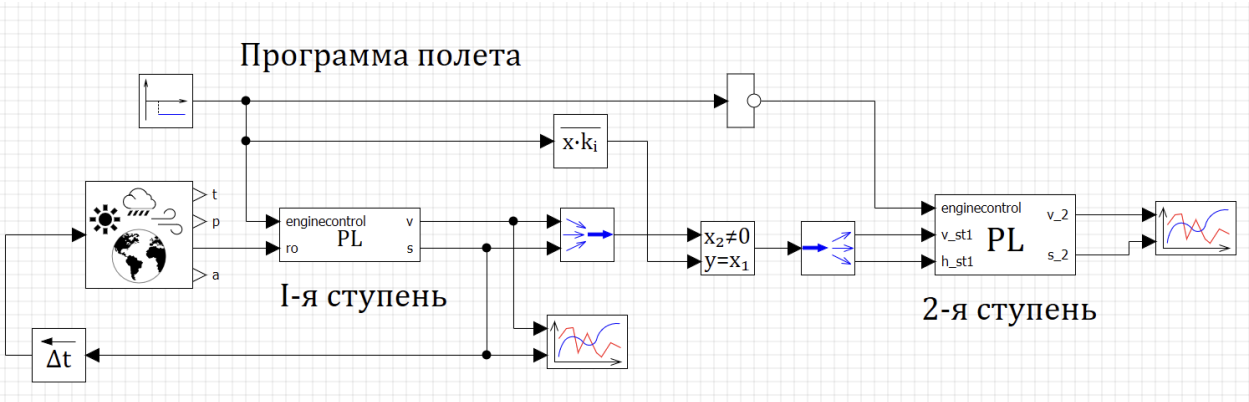
В тексте программы внесем следующие изменения:

1) В параметрах ракеты зададим данные по второй ступени
2) Для учета скорости и положения будем учитывать набранные первой ступенью скорость и высоту.
3) Пока двигатель не включен (enginecontrol = 0), ускорение и скорость также равны 0.
4) Как только двигатель включается (enginecontrol = 1) начинается расчет движения второй ступени.
5) В качестве результата отдается суммы скоростей и высоты первой ступени и второй.

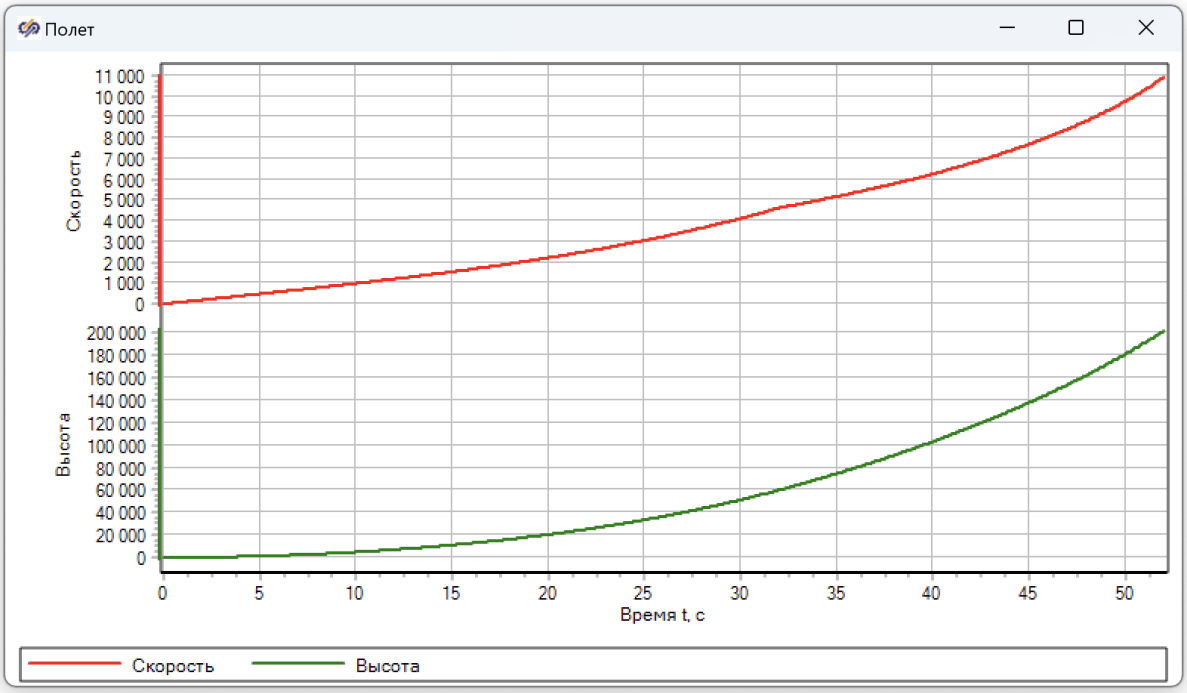
```
const
P = 10000, //полезная нагрузка
u = 5000, //скорость истечения
m_rocket = 5000,//масса второй ступени
M_fuel = 40000, //масса топлива второй ступений
mu = 2000, //расход топлива
g = 9.81, // ускорении свободного падения
end;

var muc = 2000; //расход топлива контролируемый

init //начальное состояние
s = 0, // высота в начале расчета
v = 0, // скорость на старте
M = P+m_rocket+M_fuel;  // масса на старте

muc = mu*enginecontrol//расход с учетом управления

F=muc*u; //сила тяги с учетом управления
a = (F-g*M)/M; //ускорение
v' = a*enginecontrol;  // производная скорости с учетом управления
v_2 = v + v_st1//скорость с учетом набранной скорости ступени
s' = v_2*enginecontrol; // производная высоты с учетом управления
M'= -muc; //производная массы
s_2 = s + h_st1; //высота с учетом наброной высоты первой ступени

output v_2,s_2;
```

Теперь когда вторая ступень содержит в себе все необходимое для старта займемся программой полета. У нас сейчас все очень просто первые 32 секунды работает двигатель первой ступени, затем он выключается и включается двигатель второй ступени. Такую программу можно реализовать с помощью блока ступенька, который выдает 1 в течение 32 первых секунд и 0 после.

Для первой ступени мы используем это сигнал напрямую, для второй мы его инвертируем, тогда во второй ступени первые 32 секунды полета будет 0, потом 1. Далее нам нужно запомнить высоту и скорость на которой происходит переключение режимов и передать их из первой ступени во вторую, для этого у нас есть блок запоминания сигнала. Управлять запоминанием мы будем той же ступенькой что и двигателем. Пока сигнал 1, передаются высота и скорость, как только сигнал 0, набранные скорость и высота запоминаются. И используются во второй ступени. Схема расчет достаточна проста:



Если запустить на расчет, то мы увидим, что, что вторая ступень достигает первой космической скорости и выводит спутник на орбиту!



Но самое интересное мы увидим, на графике первой ступени, после отключения двигателя ее скорость замедляется, хотя она летит по инерции. А значит мы можем эту модель использовать для расчета посадки первой ступени, как ракеты Илона нашего Маска

Вернёмся к рисункам 1 и 4. Знаете, на что они похожи? Да-да, на профиль монумента покорителям космоса, установленного в Москве у ВДНХ. Бесспорно, что архитекторы и конструкторы этого памятника имели ввиду графики изменения скорости полета ракеты во времени. И, конечно, кадры из мультфильма «Полёт на Луну».

Рис. 5. Монумент покорителям космоса

Конечно, траектория полета ракеты, выводящей на орбиту ИСЗ, далеко не вертикальная прямая, как это было принято в наших расчетах. Но их можно усложнить – ввести, например, два дифференциальных уравнения, с балансом сил по вертикали и горизонтали.

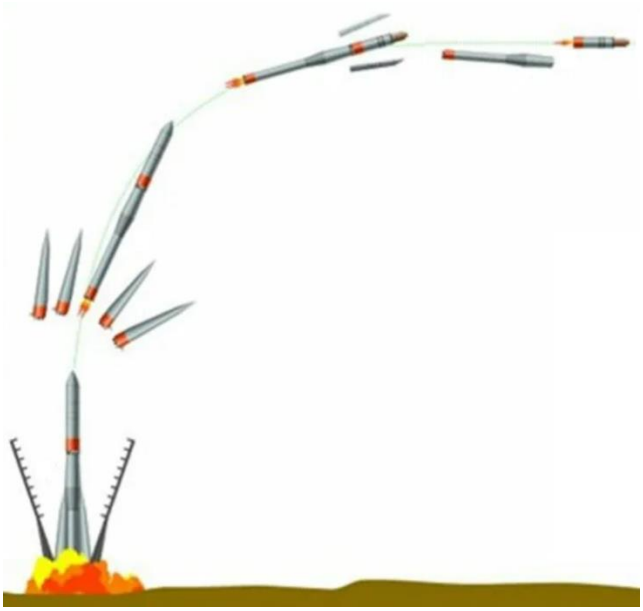


Рис. 6. Примерная траекторий вывода ИСЗ на орбиту

На сайте пользователей SMath[5] обсуждалась задача о полете ракеты. Там, в частности, рассматривалось аналитическое решение с выводом формулы Циолковского и другие интересные моменты – например, разная трактовка второго закона Ньютона применительно к телу с переменной массой (масса, умноженная на первую производную скорости по времени или первая производная произведения массы тела на его скорость – на импульс тела). Там же можно скачать решения, показанные выше.

Два послесловия.

На рисунках 2 и 3, как было уже отмечено, приведены частные случаи уравнения Мещерского[6]. Дифференциальное уравнение движения тела переменной массы, которое И.В. Мещерский вывел и опубликовал ещё в 1897 году, спустя 31 год было (по незнанию трудов Мещерского) заново выведено итальянским математиком Туллио Леви-Чивита. К сожалению, как это уже не раз бывало, о русском первооткрывателе на Западе забыли, и это уравнение стали называть уравнением Леви-Чивита.

И второе.

[5] https://en.smath.com/forum/yaf_poststt24863_Error-by-ODEs-solution.aspx и https://en.smath.com/forum/yaf_postsm84488_Euler-method-without-for.aspx
[6] https://ru.wikipedia.org/wiki/Уравнение_Мещерского

Один из авторов статьи, будучи проездом Базеле, специально отправился в пригород этого швейцарского города – в городок Риен (Riehen) и отыскал там дом, где провел детские годы великий Эйлер. На доме весела мемориальная табличка – см. рис. 7.

Рис. 7. Мемориальная табличка на доме Эйлера

Занимательная и поучительная деталь. На затылке барельефа Эйлера довольно чётко просматривается профиль льва. Суть этой детали не удалось найти при поиске в интернете и других источниках. Предположение первое – здесь зашифровано имя Эйлера – Leonhard – стойкий лев. Второй вариант – именно Эйлера можно назвать царем математики. А лев – это царь зверей.

Задание читателям.

Рассчитать полёт трёхступенчатой ракеты.