**Подробное описание автопилота выхода на орбиту (orbit.py)**

1. **Первый этап. Подготовка к взлету.**

### 

### **conn = krpc.connect()**

* Устанавливаем соединение с сервером **kRPC**, который является интерфейсом между Python и KSP.
* Это позволяет управлять кораблём и получать телеметрию из игры в реальном времени.

### **vessel = conn.space\_center.active\_vessel**

* Получает ссылку на активный космический корабль.

### **control = vessel.control**

* Ссылка на систему управления кораблём.
* Через control можно активировать ступени, регулировать тягу, изменять направление полёта и выполнять другие действия.

### **ap = vessel.auto\_pilot**

* Ссылка на автопилот корабля.
* Позволяет задавать направление (угол наклона, курс) и стабилизировать полёт без ручного управления.

### **ut\_start = conn.space\_center.ut и ut\_now = conn.space\_center.ut**

* Переменные для хранения текущего времени в игре (**Universal Time**).
* ut\_start фиксирует начальный момент времени, когда начинается работа программы.
* ut\_now используем для отслеживания времени в процессе выполнения полёта.

### **ap.target\_pitch\_and\_heading(90, 90)**

* Устанавливает начальный угол наклона и курс:
  + **Pitch (угол наклона):** 90° — вертикальный взлёт.
  + **Heading (курс):** 90° — направление на восток.

### **ap.engage()**

* Активирует автопилот, который будет поддерживать заданный угол наклона и курс.

### **Создание потоков данных**

#### **ut = conn.add\_stream(getattr, conn.space\_center, 'ut')**

* Поток, возвращающий текущее время в игре (**Universal Time**).
* Используется для расчёта продолжительности полёта или тайминга манёвров.

#### **stage\_5\_resources = vessel.resources\_in\_decouple\_stage(stage=5, cumulative=False)**

* Получает доступ к ресурсам пятой ступени (ускорители) корабля.
* Аргументы:
  + stage=5 — номер ступени.
  + cumulative=False — ресурсы учитываются только для указанной ступени, без объединения с другими.

#### **srb\_fuel = conn.add\_stream(stage\_5\_resources.amount, 'SolidFuel')**

* Создаёт поток для отслеживания оставшегося твёрдотопливного топлива в пятой ступени (ускорители).

#### **altitude = conn.add\_stream(getattr, vessel.flight(), 'mean\_altitude')**

* Поток для отслеживания высоты над уровнем моря.

#### **apoapsis = conn.add\_stream(getattr, vessel.orbit, 'apoapsis\_altitude')**

* Поток для отслеживания высоты апоцентра орбиты.
* Апоцентр — самая высокая точка орбиты.

#### **periapsis = conn.add\_stream(getattr, vessel.orbit, 'periapsis\_altitude')**

* Поток для отслеживания высоты перицентра орбиты.
* Перицентр — самая низкая точка орбиты.

#### **pitch = conn.add\_stream(getattr, vessel.flight(), 'pitch')**

* Поток для получения текущего угла наклона корабля относительно горизонта.

### 

### 

1. **Второй этап. Запуск ускорителей.**

### 

#### **Обратный отсчёт**

print("3...")

time.sleep(0.5)

print("2...")

time.sleep(0.5)

print("1...")

time.sleep(0.5)

* Выводит на экран последовательный отсчёт перед запуском ракеты.
* Использует time.sleep(0.5) для задержки выполнения программы на 0.5 секунды между сообщениями.

#### **Активация ускорителей**

print("Start!")

control.activate\_next\_stage()

start\_time = time.time()

* **control.activate\_next\_stage()**: Запускает ускорители.

### **Функция изменения угла наклона (сердце автопилота)**

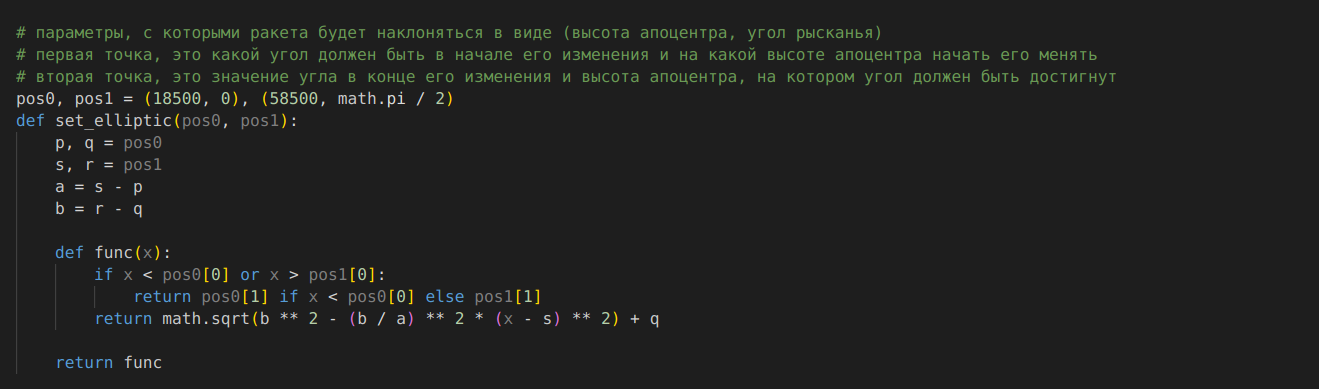
Ракета сначала летит строго вертикально (90°), но должна постепенно ложиться на горизонтальный полёт. Этот процесс управляется гравитационным поворотом.

#### **Параметры гравитационного поворота**

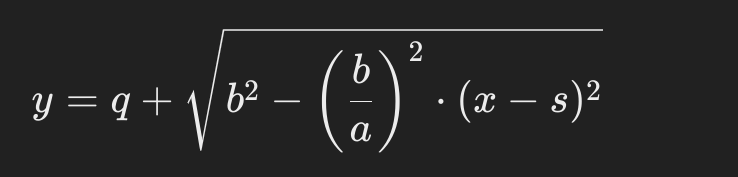
**pos0, pos1 = (18500, 0), (58500, math.pi / 2)**

* **pos0 = (18500, 0)**:
  + При апоцентре (высота самой дальней точки траектории) **18,500 м**, ракета начинает изменять угол наклона.
  + Угол в этот момент — **0°** (полет строго вертикально).
* **pos1 = (58500, math.pi / 2)**:
  + При апоцентре **58,500 м**, угол наклона ракеты должен быть **90°** (горизонтальный полёт).

#### **Определение функции изменения угла**



1. **p, q = pos0 и s, r = pos1:**
   * Переменные для высоты (p и s) и углов (q и r).
2. **a = s - p и b = r - q:**
   * Расстояния между начальной и конечной точками по осям высоты и угла.
3. **if x < pos0[0] or x > pos1[0]:**
   * Если текущая высота x находится вне заданного диапазона [p, s]:
     + Вернуть угол начальной точки q для высоты ниже pos0[0].
     + Вернуть угол конечной точки r для высоты выше pos1[0].
4. **return math.sqrt(...)**:  
   * Если x находится в диапазоне [p, s], угол рассчитывается по эллиптической формуле:



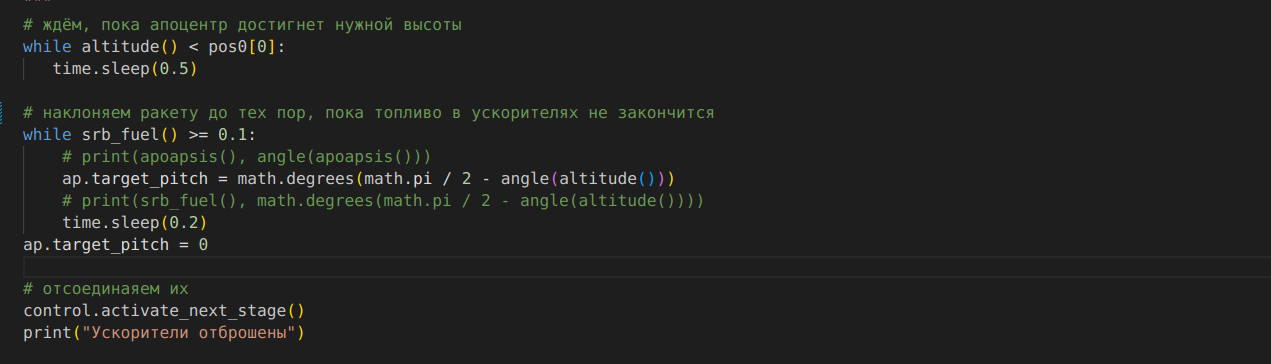
* + - Гарантирует плавное изменение угла от начального значения до конечного.

#### **Инициализация функции**

angle = set\_elliptic(pos0, pos1)

* angle — это теперь функция, которая принимает высоту апоцентра и возвращает угол наклона ракеты.

Пример использования:  
 print(angle(20000)) # Угол наклона при апоцентре 20,000 м



### **3 Этап. Доорбитальный полет.**

#### **Задачи:**

1. Плавно изменить угол наклона ракеты, переходя к горизонтальному полёту (гравитационный поворот).
2. Отбросить ускорители после их израсходования.

#### **Ждём достижения нужной высоты апоцентра**

while altitude() < pos0[0]:

time.sleep(0.5)

* Цикл проверяет текущую высоту ракеты с помощью altitude().
* **Цель:** дождаться, пока высота апоцентра достигнет **18500 м** (значение pos0[0]).
* Использует time.sleep(0.5) чтобы KSP не завис.

#### **Наклон ракеты**

while srb\_fuel() >= 0.1:

ap.target\_pitch = math.degrees(math.pi / 2 - angle(altitude()))

time.sleep(0.2)

* Пока у ускорителей остаётся хотя бы 10% топлива:
  + **angle(altitude())**: Рассчитывает целевой угол наклона на основе текущей высоты ракеты.
  + **math.pi / 2 - angle(...)**: Преобразует значение угла в наклон ракеты.
  + **ap.target\_pitch = ...**: Автопилот меняет угол ракеты на рассчитанный.
* Задержка time.sleep(0.2) снижает нагрузку на KSP и ПК.

#### **Устанавливаем горизонтальный полёт**

ap.target\_pitch = 0

* После окончания топлива у ускорителей:
  + Устанавливает угол наклона на **0°**, переходя к горизонтальному полёту.

#### **Отделяем ускорители**

control.activate\_next\_stage()

print("Ускорители отброшены")

* **control.activate\_next\_stage()**: Выполняет отделение ускорителей.
* Сообщение **"Ускорители отброшены"** выводится в терминал для информирования.

#### **Функция для расчёта манёвра (гомановский переход)**

**Описание параметров:**

1. **mu**: Стандартный гравитационный параметр тела.
   * Определяется как произведение гравитационной постоянной на массу тела.
2. **r1**: Радиус начальной орбиты (например, перицентр текущей орбиты).
3. **r2**: Радиус целевой орбиты (например, апоцентр переходной орбиты).

**Рассчитывает:**

**dv1 - На такое значение нужно увеличить скорость, будучи на круговой орбите r1**

**dv2 - на Такое значение нужно увеличить скорость, будучи на переходной траектории гомана (высота апоцентра это r2)**

#### **Применение функции и расчёт манёвра**

mu = vessel.orbit.body.gravitational\_parameter

delta\_v = gam(mu, vessel.orbit.periapsis, vessel.orbit.apoapsis)[1]

1. **mu**: Вычисляется из гравитационного параметра текущей планеты или тела.
   * Доступен через vessel.orbit.body.gravitational\_parameter.
2. **delta\_v**: Берётся второе значение из функции test3, которое соответствует скорость для выхода на целевую орбиту.

#### **Добавление манёвра**

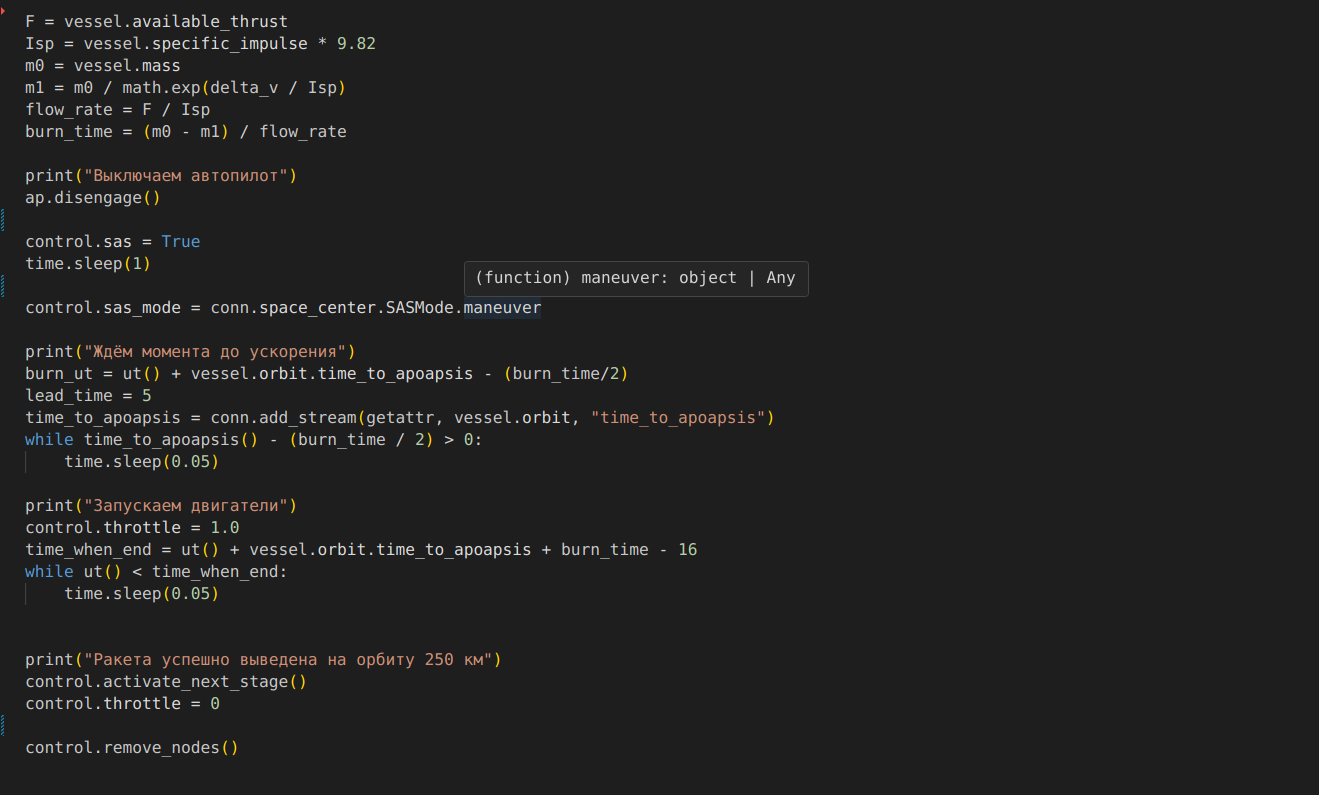
node = control.add\_node(ut() + vessel.orbit.time\_to\_apoapsis, prograde=delta\_v)

* **ut()**: Текущее универсальное время в игре.
* **vessel.orbit.time\_to\_apoapsis**: Время до апоцентра текущей орбиты.
* **control.add\_node**: Добавляет узел манёвра с указанными параметрами:
  + Время манёвра: ut() + vessel.orbit.time\_to\_apoapsis
  + Прирост скорости (prograde): prograde=delta\_v

#### **Добавление задержки**

time.sleep(1)

* Делает паузу в 1 секунду перед продолжением следующей команды.



#### **Расчёт параметров для манёвра:**

**Определение доступной тяги и других характеристик ракеты:** F = vessel.available\_thrust

* + **F**: Максимальная доступная тяга ракеты (в Ньютонах).

Isp = vessel.specific\_impulse \* 9.82

* + **Isp**: Удельный импульс двигателя в м/с. Переводится из секунд (умножая на ускорение свободного падения, 9.82).

m0 = vessel.mass

* + **m0**: Начальная масса ракеты (в кг).

m1 = m0 / math.exp(delta\_v / Isp)

* + **m1**: Окончательная масса ракеты после сжигания топлива для выполнения манёвра, рассчитывается по уравнению Циолковского:

flow\_rate = F / Isp

* + **flow\_rate**: Расход топлива (кг/с)

burn\_time = (m0 - m1) / flow\_rate

* + **burn\_time**: Время сжигания топлива (в секундах).

#### **Выключение автопилота:**

print("Выключаем автопилот")

ap.disengage()

* Отключает текущий автопилот ракеты.

#### **Переход к ручному управлению с использованием SAS:**

control.sas = True

time.sleep(1)

control.sas\_mode = conn.space\_center.SASMode.maneuver

* **control.sas = True**: Включает SAS (система автоматической стабилизации).
* **control.sas\_mode**: Устанавливает режим SAS в **maneuver** (наведение на точку манёвра).

#### **Ожидание момента для начала манёвра:**

burn\_ut = ut() + vessel.orbit.time\_to\_apoapsis - (burn\_time / 2)

lead\_time = 5

time\_to\_apoapsis=conn.add\_stream(getattr,vessel.orbit,"time\_to\_apoas”)

* **burn\_ut**: Рассчитывает точное время старта манёвра:
  + Манёвр должен начаться за половину времени сжигания топлива до апоцентра.
* **time\_to\_apoapsis**: Создаёт поток, который отслеживает время до апоцентра.

while time\_to\_apoapsis() - (burn\_time / 2) > 0:

time.sleep(0.05)

* Ожидает, пока до апоцентра не останется меньше половины времени сжигания топлива.

#### **Запуск двигателей:**

print("Запускаем двигатели")

control.throttle = 1.0

* Устанавливает полный газ (throttle=1.0)

#### **Выполнение манёвра:**

time\_when\_end = ut() + vessel.orbit.time\_to\_apoapsis + burn\_time - 16

while ut() < time\_when\_end:

time.sleep(0.05)

* **time\_when\_end**: Рассчитывает момент окончания манёвра.
  + Манёвр продолжается до конца рассчитанного времени.
* **while ut() < time\_when\_end**: Следит за текущим временем и завершает цикл, когда манёвр выполнен.

#### **Завершение манёвра:**

print("Ракета успешно выведена на орбиту 250 км")

control.activate\_next\_stage()

control.throttle = 0

* Оповещает о завершении манёвра.
* Активирует следующую ступень ракеты.
* Останавливает двигатели (throttle=0\text{throttle} = 0).

#### **Удаление узлов манёвра:**

control.remove\_nodes()

* Удаляет все узлы манёвра, чтобы не мешать дальнейшему управлению.