Aufgabe 16 - Titan, Mond von Saturn

Clemens Ibrom

May 2024

1 Die Bewegungsgleichung

Unter den in der Aufgabe beschriebenen Annahmen ist die Bewegungsgleichung (BWGL) von Titan:

$$m_{\text{Titan}}\ddot{\vec{r}} = -G \cdot \frac{M_{\text{Saturn}} \cdot m_{\text{Titan}}}{r^2} \cdot \hat{r} \tag{1}$$

hier $\hat{r}$ der normierte Abstandsvektor in Richtung Saturn. Wie im Aufgabentext beschrieben, legen wir den Saturn in den Ursprung und ignorieren seine Bewegung. Somit ist das Problem zweidimensional.

2 Runge-Kutta 4. Ordnung

Für diese Simulation benutzten wir den Runge-Kutta Algorithmus 4. Ordnung (RK4). Der Algorithmus ist eine Verbesserung von dem Euler-Verfahren. Gegegben eine gewöhnliche DGL 1. Ordnung

$$\frac{dy}{dt} = F(y,t)$$

berechnen wir die Veränderungen an mehreren, hier 4, Stützstellen. Die Zeit ist diskretisiert mit $t \in \{t_i\}_i$ und wir schrieben $y_i := y(t_i)$. Dann berechnen wir die vier Werte:

$$\begin{aligned}
k_1 &= \Delta t \cdot F(y_i, t_i) \\
k_2 &= \Delta t \cdot F(y_i + \frac{k_1}{2}, t_i + \frac{t}{2}) \\
k_3 &= \Delta t \cdot F(y_i + \frac{k_2}{2}, t_i + \frac{\Delta t}{2}) \\
k_4 &= t \cdot F(y_i + k_3, t_{i+1}).
\end{aligned}$$

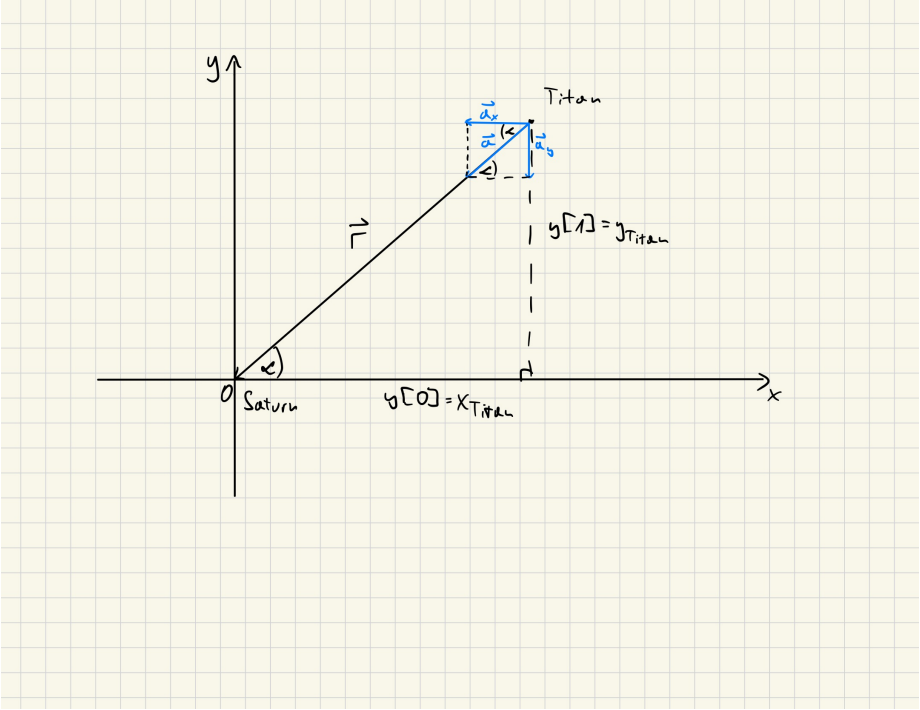
Dann ist $y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(\Delta t^5)$ die Ausgabe von einem RK4 Schritt.

3 Die Differenzialgleichung implementiert

Wie in Teil 1 beschrieben haben wir es mit einem zweidimensionalen Problem zu tun, daher schreiben wir in den Zustandsvektor nur die x, y-Koordinaten und die Gechwindigkeiten in diese Richtungen:

$$\vec{y} = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix}. \tag{2}$$

Um die BWGL in Komponenten auszurechnen muss man beachten, dass die Beschleunigung gegeben in Gl. 1 Vektoriell ist, also müssen die x und y Komponenten berechnet werden. Das kann z.B. geometrisch erfolgen, mittels ähnlicher Dreiecke:



Dann gilt

$$\sin\alpha = \frac{y[1]}{r} = \frac{a_y}{a}$$

wo wir mit $r = |\,\vec{r}\,|$ und $a = |\,\vec{a}\,| = -G \cdot \frac{M_{Saturn}}{r^2}$ die Größe $a_y = |\,\vec{a_y}\,|$ ausrechenen können, da die y-Koordinate von Titan auch bekannt ist. Analog gehen wir für

die x-Komponente der Bechleunigung vor und bekommen die Beiden Wert:

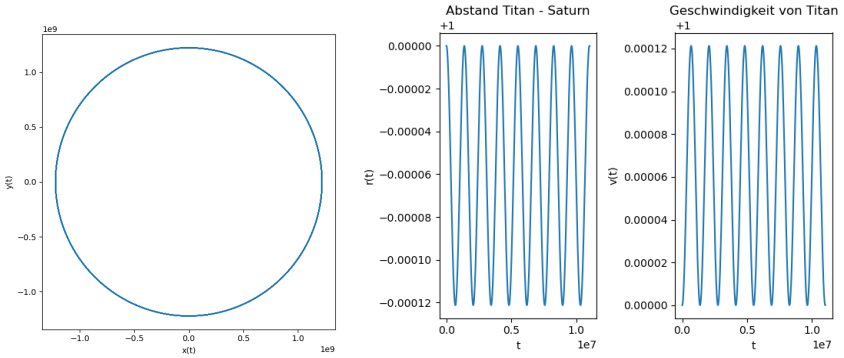
$$a_x = -G \cdot \frac{M_{Saturn} \cdot y[0]}{r^3}$$
$$a_y = -G \cdot \frac{M_{Saturn} \cdot y[1]}{r^3}.$$

Somit ist der Output der ODE Funktion

$$dy = \begin{pmatrix} v_x \\ v_y \\ a_x \\ a_y \end{pmatrix}. \tag{3}$$

4 Erste Simulation

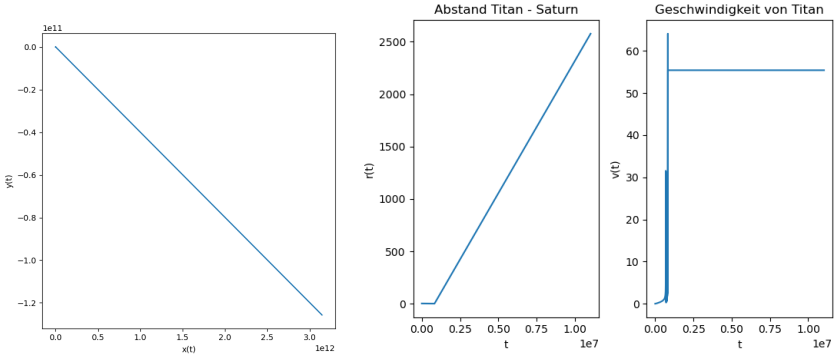
In Aufgabenteil 3 lassenn wir die erste Simulation der Bewegung von Titan laufen. Hierfür starten wir im Perihel und integrieren den Zustandsvektor mit dem RK4 Stepper über 8 Orbitalperioden mit Schrittweite 10^{-3} Mal Periodendauer von Titan. Wie erwartet bekommen wir als Teajektorie einen fast perfekten Kreis:



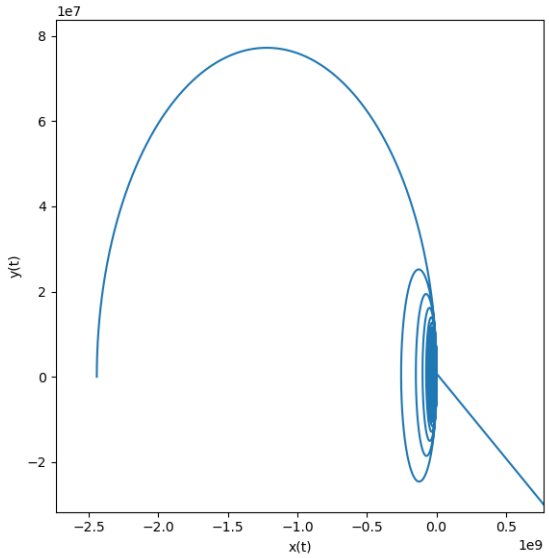
Wie man an der y-Achse ablesen kann, ist die Variation vom Abstand und von der Geschwindigkeit sehr klein. Das, und der Trajektorie Plot lassen darauf schliessen, dass diese Simulation ziemlich genau war und der Theoretischen Erwartung entspricht.

5 Start im Aphelion

Nun wollen wir eine Exzentrizität von fast 1 annehmen und unsere Simulation im Aphelion starten. Zu erst machen wir das mit festen Zeitschritt von 10^{-5} mal die Periodendauer. Das resultiert in einer unstabilen Simulation und der Mond wird aus der Orbit geschleudert. Die Plots sind:



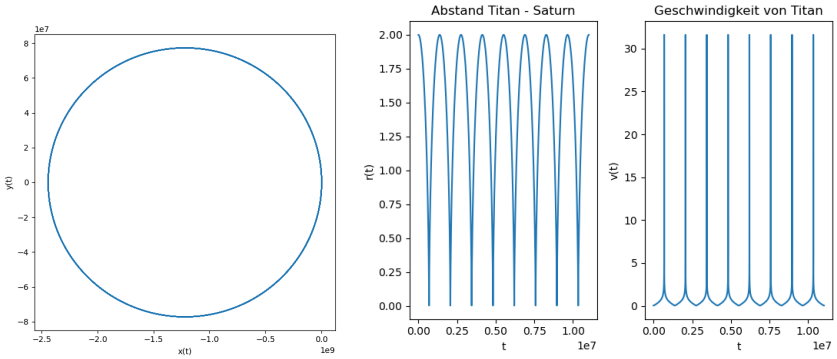
Wie man an den Plots erkennen kann wird der Titan aus der Orbit geschleudert. Dies geschieht weil bei festem Zeitschritt der Fehler im Integrationsschritt zu groß wird was zur destabilisierung führt. Wenn man in dem Trajektorieplot reinzoomed sieht man wie an einer Stelle die Ellipsen abrupt enger werden bis der Mond wegschießt.



Die Simulation hat 800001 Schrite gebraucht.

5.1 Adaptiver Zeitschritt

Um der destabilisierung vorzubeugen führen wir den adaptiven Zeitschritt ein. Dazu Integrieren wir ein Mal mit dem ganzen Zeitschritt und gleichzeitig zwei Mal mit dem halben Zeitschritt. Dann berechnen wir den Betrag vom unterschied der Ergebnisse. Diese Größe benutzten wir um den Zeitschritt zu korrigieren, wie in der aufgabe beschrieben. Dadurch wird jeder Schritt genauer, also die Fehler kleinere, und die Simulation wird Stabil. Hier die Ergebnisse:



Die Simulation mit adaptiven Zeitschritt hat nur 6870 Schritte gebraucht.