Indhold

1 Indledning og problemstilling

Simulation af mekanik har været en del af videospil, siden fysikeren William Higinbotham programmerede et af de første videospil, "Tennis for Two" i 1958. Simulation af mekaniske systemer anvendes ofte at gøre videospil mere realistiske og underholdende. I denne opgave er målet at simulere bevægelsen i en svævebane. Det er relevant i flere forskellige computerspil, og noget kan se kunstigt ud, hvis den bagvedliggende fysiske model ikke er god. Man ønsker som altid i computergrafik at finde en god balance mellem realisme og performance.

Hvilke udfordringer er der i at modellere en realistisk bevægelse i en svævebane, og implementere den i et computerprogram?

2 Problemformulering

- Der redegøres for relevante formalismer for mekanik og bevægelsesligningerne opstilles for bevægelsen i svævebaner, idet der trinvist tages hensyn til så mange parametre som muligt.
- Der implementeres et program, der kan visualisere bevægelsen, givet parametrene ovenfor, og programmet anvendes til at undersøge sammenhængen mellem hvor realistisk bevægelsen modelleres og køretiden af programmet.
- Der diskuteres og laves en vurdering af, hvad der vil være en passende model for en svævebane i et computerspil.

3 Metode

Jeg har anvendt *Lagranges formalisme* for mekanik, da man let kan arbejde med generaliserede koordinater udover det kartesiske. Jeg har brugt Eulers metode som numerisk metode, da den er simpelt at bruge.

Jeg har anvendt *pygame* og *matplotlib* til at visualisere bevægelsen og teste køretiden. De er forholdsvis simpelt og kan bruges i sammenhæng med numpy og pandas for øvet funktionalitet.

Jeg har primæt anvendt John R. Taylors bog »Classical Mechanics« som kilde for definitioner og ligninger i Lagranges formalisme og J. Kim Vandivers lektioner som kilde for ligninger for generaliserede kræfter, da kildematerialerne er præcist i de tekniske detaljer, men er også begyndervenligt.

Desuden har jeg primært refereret til de fine dokumentationer af de fornævnte moduler som kilde, men også anvendt et række forum-posts på nettet for at løse nogle specifikke problemer.

4 Redegørelse af Lagranges formalisme

Lagranges formalisme, opkaldt efter Joseph-Louis Lagrange (1736-1813)[1], er en anden måde at nå frem til de samme bevægelsesligninger, som man kan gøre gennem Newtons formalisme.

Fordelen med Lagranges formalisme er, at man gennem den kan få automatisk konverteret de kartesiske koordinater om til sine ønskede generaliserede koordinater, som tager hensyn til et problems begrænsninger (ellers kaldet for »*constraints*« på engelsk).[2]

De generaliserede koordinater kan være mere nyttigt at bruge. Ønsker man at for eksempel beskrive positionen af et simpelt pendul, vil man hellere gøre det med en vinkel i stedet for bruge de kartesiske koordinater.

Lagranges formalisme er dog mere abstrakt end Netwons formalisme, og jeg vil derfor benytte mig af et eksempel til at forklare den, hvor eksemplet er sædvanligvis et simpelt pendul.

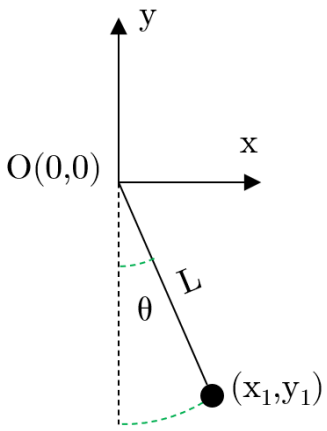
Jeg definerer et pendul med masse m for at være bundet til origo med en stiv og masseløs snor med længde L.

Snoren danner en vinkel θ med y-aksen.

Pendulets position, (x_1, y_1), kan beskrives af θ på følgende måde:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} L \cdot sin\theta \\ -L \cdot cos\theta \end{pmatrix}$$

, hvor x_1, y_1 og θ afhænger af tiden, t, men det opskrives ikke at gøre det mere læseligt.



Pendulets hastighed, $(\dot{x}_1, \dot{y}_1)$, findes ved at differentiere positionsudtrykket med hensyn til tiden t:

$$\begin{pmatrix} \dot{x_1} \\ \dot{y}_1 \end{pmatrix} = \begin{pmatrix} L \cdot cos\theta\dot{\theta} \\ L \cdot sin\theta\dot{\theta} \end{pmatrix}$$

[1] Wikipedia bidragsydere, "Joseph-Louis Lagrange," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Joseph-Louis_Lagrange&oldid=1121816812 (lokaliseret d. 17 december 2022).

[2] J. R. Taylor: »Classical Mechanics«, University Science Books: California, s. 243-244.

Da jeg har et udtryk for position, kan jeg finde pendulets potentielle energi, U:

$$\begin{aligned} U &= mgh \\ &= mg(L - L \cdot cos\theta) \\ &= mgL(1 - cos\theta) \end{aligned}$$

Da jeg har et udtryk for hastighed, kan jeg finde pendulets kinetiske energi, T:

$$\begin{aligned} T &= \frac{1}{2}mv^2 \\ &= \frac{1}{2}m(\dot{x}^2 + \dot{y}^2) \\ &= \frac{1}{2}m((L \cdot cos\theta\dot{\theta})^2 + (L \cdot sin\theta\dot{\theta})^2) \\ &= \frac{1}{2}m(L^2cos^2\theta\dot{\theta}^2 + L^2sin^2\theta\dot{\theta}^2) \\ &= \frac{1}{2}m(L^2\dot{\theta}^2(sin^2\theta + cos^2\theta)) \\ &= \frac{1}{2}mL^2\dot{\theta}^2 \end{aligned}$$

, hvor $sin^2\theta + cos^2\theta = 1$ er en trigonometrisk identitet.[3]
Grunden til, at jeg har udledt pendulets potentielle og kinetiske energi, er for at finde det simple penduls *Lagrange*-funktion.

Lagrange-funktionen er en mængde der karakteriserer tilstanden af et system.[4] Den defineres som differensen mellem den kinetiske energi og den potentielle energi.[5] Man kan nemlig opskrive ligningen for Lagrange-funktionen som følgende:

$$\mathcal{L} = T - U \text{ (Langrange-funktionen)} \tag{1}$$

For det simple pendul vil den være:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}mL^2\dot{\theta}^2 - mgL(1 - cos\theta) \\ &= \frac{1}{2}mL^2\dot{\theta}^2 - mgL + mgLcos\theta \\ &= \frac{1}{2}mL^2\dot{\theta}^2 + mgLcos\theta - mgL \end{aligned}$$

[3] University of Liverpool: »Trigonometric Identities«, https://www.liverpool.ac.uk/∼maryrees/homepagemath191/trigid.pdf (lokaliseret d. 17 december 2022).

[4] Britannica, T. Redaktører of Encyclopaedia. "Lagrangian function." Encyclopedia Britannica, https://www.britannica.com/science/Lagrangian-function (lokaliseret d. 17 december 2022).

[5] J. R. Taylor: »Classical Mechanics«, University Science Books: California, s. 238

Grunden til, at *Lagrange*-funktionen er så nyttigt, er fordi naturen helt op fra den astrofysiske skala ned til kvanteskala, viser sig at følge hvad der er kaldt for *Hamiltons princip.*[6]

Hamiltons princip

Den aktuelle bane, som en partikel følger mellem to punkter P_1 og P_2 i en given tidsinterval $[t_1; t_2]$, er således, virkningsintergralet

$$S = \int_{t_1}^{t_2} \mathcal{L} dt$$

er *stationær*, når den er taget ad den aktuelle bane.

Grunden til, at der er tale om en aktuel bane, er fordi, man betragter alle de baner en partikel kan tage mellem to punkter og finder den bane, hvor virkningsintegralet er stationær. Det er stationær på den måde, at en uendelig lille variation fra partiklens aktuelle bane ændrer ikke værdien af S.[7]

Leonhard Euler (1707-1783)[8] har sammen med Joseph-Louis Lagrange været med til at finde en ligning, der kan finde den bane, hvor S 12er stationær. Ligningen, der er opkaldt efter dem som *Euler-Lagrange* ligningen, kan opskrives som følgende med hensyn til lagrange-funktionen:

$$\frac{\partial \mathcal{L}}{\partial q_i} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) \text{ (Euler-Lagrange ligningen)} \tag{2}$$

, hvor $\mathcal{L}$ er lagrange-funktionen (1), t er tiden, i går fra 1 til n, hvor n er antallet af koordinater, og q_i er den i'te koordinat.

Anvender jeg Euler-Lagrange-ligningen (2) til eksemplet med det simple pendul, har jeg:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) \quad \text{, hvor:} \quad \mathcal{L} = \frac{1}{2}mL^2\dot{\theta}^2 + mgLcos\theta - mgL$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = -mgLsin\theta$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(mL^2\dot{\theta}\right) = mL^2\ddot{\theta}$$

[6] Wikipedia bidragsydere, "Hamilton's principle," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Hamilton%27s_principle&oldid=1117662115 (lokaliseret d. 17 december 2022).

[7] Taylor, s. 218-220

[8] Wikipedia bidragsydere, "Leonhard Euler," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Leonhard_Euler&oldid=1127824460 (lokaliseret d. 17 december 2022).

Og jeg kan omskrive Euler-Lagrange-ligning for det simple pendul som følgende:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \theta} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) \\
-mgLsin\theta &= mL^2\ddot{\theta} \\
mL^2\ddot{\theta} + mgLsin\theta &= 0 \\
\ddot{\theta} + \frac{g}{L}sin\theta &= 0
\end{aligned}$$

Laver man det såkaldte »lille vinkels approksimation«[9] og sætter $sin\theta \approx \theta$, kan bevægelsesligningen ovenfor løses til:

$$\theta(t) = \theta_0 \cdot cos\left(\sqrt{\frac{g}{L}} \cdot t\right)$$

Man kan vise, at Euler-Lagrange-ligningen (2) er en form for generaliseret version af Newtons 2. lov. Da den kinetiske energi afhænger af hastigheden i x- og y-retningen $\dot{x}$ og $\dot{y}$, og den potentielle energi afhænger af positionen i x- og y-retningen[10][11], kan man argumentere for følgende i y-retningen:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \theta} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) \\
\frac{\partial(T(\dot{x},\dot{y}) - U(x,y))}{\partial y} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial(T(\dot{x},\dot{y}) - U(x,y))}{\partial \dot{y}}\right) \\
\Leftrightarrow \frac{\partial(mgy)}{\partial y} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial(\frac{1}{2}m\dot{y}^2)}{\partial \dot{y}}\right) \\
\Leftrightarrow F_y &= m\ddot{y}
\end{aligned}$$

, og på tilsvarende måde i x-retningen.
Hvis man arbejder med ligning (2) er man dog kun begrænset til de konservative kræfter, der tilføres gennem Lagrange-funktionen (1). Hvis man ønsker at inddrage ikke-konservative kræfter, kan man heldigvis gøre det ved at tilføre et generaliseret kraft-led, Q_i, i Lagrange-ligningen som følgende:

$$\frac{\partial \mathcal{L}}{\partial q_i} + Q_i = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) \tag{3}$$

, hvor $\mathcal{L}$ er lagrange-funktionen, t er tiden, i går fra 1 til n, hvor n er antallet af koordinater, q_i er den i'te koordinat, og Q_i er et generaliseret kraft-led forbundet med q_i.

[9] Wikipedia bidragsydere, "Lille vinkel", Wikipedia, Den frie encyklopædi., https://da.wikipedia.org/w/index.php?title=Lille_vinkel&oldid=11105739 (lokaliseret d. 17 december 2022).

[10] Hvis man forestiller sig at rotere y-aksen, så partiklens beliggenhed projiceret i tyngekraftens retning er udtrykt af både x og y.

[11] Bemærk dog, at den potentielle energi altid ikke kun afhænger af positionen. Under et homogent tyngdefelt som jeg arbejder med gør den det.

Måden den generaliseret kraft-led findes er ved at betragte det »virtuelle« arbejde, som de ydre kræfter udfører ved at skubbe (mht. mit eksempel) pendulet i θ-aksen en lille $\delta\theta$.[12] Ved at gøre det (mht. enhver partikel i q_i-retning), kan man udlede følgende ligning, der kan bruges til at finde enhver generaliseret kraft[13][14]:

$$Q_i = \sum_{j=1}^{m} \vec{F_j} \cdot \frac{\partial \vec{r_j}}{\partial q_i} \text{ (Generaliseret Kraft)} \tag{4}$$

, hvor j går fra 1 til m, m er antallet af ydre kræfter i systemet, F_j er en ydre kraft og r_j er strækningsvektoren til partiklerne, hvor de ydre kræfter er påført.

Som eksempel, hvis jeg påfører pendulet, hvilkets position kan beskrives af $\Big(x_1, y_1\Big) = \Big(L \cdot sin\theta, -L \cdot cos\theta\Big)$, med en ydre kraft i x-retningen, $\vec{F_1} = \Big(-a, 0\Big)$, vil den generaliseret kraft-led være:

$$\begin{aligned} Q_1 &= \begin{pmatrix} -a \\ 0 \end{pmatrix} \cdot \frac{\partial \Big(\Big(L \cdot sin\theta, -L \cdot cos\theta\Big)\Big)}{\partial \theta} \\ &= \begin{pmatrix} -a \\ 0 \end{pmatrix} \cdot \begin{pmatrix} L \cdot cos\theta \\ L \cdot sin\theta \end{pmatrix} \\ &= -a \cdot L \cdot cos\theta - 0 \cdot L \cdot sin\theta \\ &= -a \cdot L \cdot cos\theta \end{aligned}$$

Og jeg får følgende bevægelsesligning:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta} + Q_1 &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) \\ \Leftrightarrow -mgLsin\theta - a \cdot L \cdot cos\theta &= mL^2\ddot{\theta} \\ \Leftrightarrow \ddot{\theta} + \frac{g}{L}sin\theta + \frac{a}{mL} \cdot cos\theta &= 0 \end{aligned}$$

Svære differentialligninger som ovenfor kan også løses numeriske metoder.

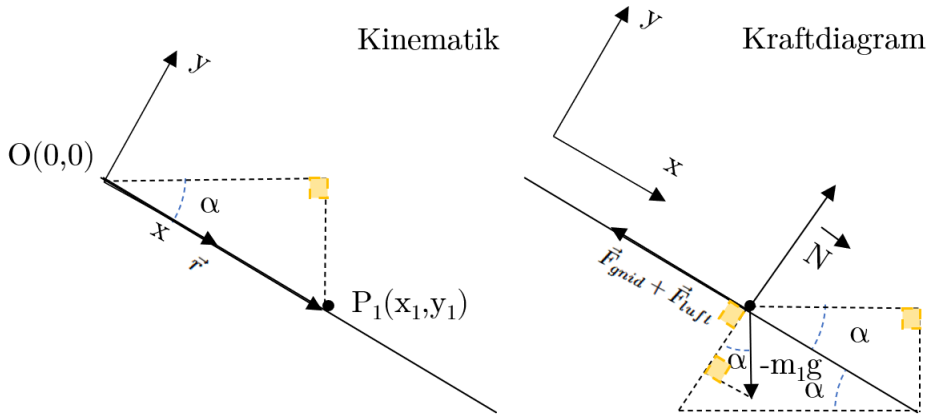
[12] J. Kim Vandiver, 2011b: »16. Kinematic Approach to Finding Generalized Forces«, MIT OpenCourseWare, 28:46-34:09 https://youtu.be/lFedznDnPZc?t=1726(lokaliseret d. 17 december 2022).

[13] J. Kim Vandiver, 2011a: »R9. Generalized Forces«, MIT OpenCourseWare https://youtu.be/QadsG49DY3M (lokaliseret 17/12/2022)

[14] Wikipedia bidragsydere, "Generalized forces," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Generalized_forces&oldid=798161881 (lokaliseret d. 17 december 2022)..

5 Bevægelsesligningerne for bevægelse i svævebaner

5.1 Det simple model



I forbindelse med min problemformulering er jeg interesseret i at sammenligne ligende »modeller« for bevægelse i svævebaner. Derfor vil jeg udvikle denne model som indledning til min mere kompleks model.

Jeg antager, at rebet er statisk dvs. den bukker sig ikke, og at origoen ligger ved rebets start.

Vektor $\vec{r} = (x_1, y_1)$ går fra origo til P_1, hvor der ligger en punktmasse med masse m. Punktmassen repræsenterer rytteren i en svævebane.

Vektor $\vec{r}$ danner en vinkel α med jorden, og følgende definitioner slås fast:

- $g = 9.81 \frac{\mathrm{m}}{\mathrm{s}^2}$
- $R \in \mathbb{R}$
- $\alpha \in \left\{ x \in \mathbb{R} : 0 \leq x \leq \pi \right\}$

Jeg antager, at punktmassen er påvirket af fire kræfter, normalkraften, tyngdekraften, gnidningskraften og luftmodstandskraften, hvor de to sidstnævnte er ikke-konservative og vil tilføres gennem et generaliseret kraft-led.

Systemet har en frihedsgrad og kan beskrives af koordinaten R. Som før, skriver jeg ikke x_1, y_1 Rs afhængighed af tiden, t, for at gøre det mere læseligt.

Positionen beskrives gennem R og differentieres, så hastigheden findes:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} R \\ 0 \end{pmatrix} \Leftrightarrow \begin{pmatrix} \dot{x}_1 \\ \dot{y}_1 \end{pmatrix} = \begin{pmatrix} \dot{R} \\ 0 \end{pmatrix}$$

Den potentielle og den kintetiske energi af systemet bestemmes, hvorefter Lagrange-funktionen (1) bestemmes:

$$\begin{aligned} U &= mgh = -mgsin\alpha R \\ T &= \frac{1}{2}mv^2 = \frac{1}{2}m\dot{R}^2 \\ \mathcal{L} &= T - U = \frac{1}{2}m\dot{R}^2 + mgsin\alpha R \end{aligned}$$

Hvis man antager, de ydre kræfter *ikke* påføres, kan man gennem Euler-Lagrange-ligningen (2) finde frem til følgende bevægelsesligning:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial R} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{R}}\right) \\ \Leftrightarrow mgsin\alpha &= m\dot{\ddot{R}} \\ \Leftrightarrow \ddot{R} &= gsin\alpha \end{aligned}$$

for hvad der er findes en analytisk løsning for:

$$\boxed{\begin{aligned} &\ddot{R} = gsin\alpha \\ &\dot{R} = tgsin\alpha + \dot{R}_0 \\ &R = \tfrac{1}{2}gsin\alpha \cdot t^2 + \dot{R}_0 \cdot t + R_0 \end{aligned}}$$

For at inddrage de ydre kræfter skal jeg bestemme den generaliseret kraft. For at gøre det, skal jeg bestemme gnidningskraften, der afhænger af normalkraften. Ved at betragte kraftdiagrammet ovenpå, hvor man også ved, at punktmassen ikke bevæger sig i y-retningen, kan man finde normalkraften på følgende måde:

$$\sum F_{resy} = 0 = \vec{N} - mgcos\alpha \Leftrightarrow \vec{N} = mgcos\alpha$$

Jeg ved, at både gnidningskraften og luftmodstandskræften påføres modsatrettet punktmassens bevægelse, så

$$\begin{aligned} \vec{F}_{luft} &= -k|\vec{v}|^2\vec{e} \wedge \vec{e} = \frac{\vec{v}}{|\vec{v}|} \\ \Leftrightarrow \vec{F}_{luft} &= -k|\vec{v}|\vec{v} = -k|\dot{R}^2 + 0^2| \begin{pmatrix} \dot{R} \\ 0 \end{pmatrix} = -k\dot{R} \begin{pmatrix} \dot{R} \\ 0 \end{pmatrix} \end{aligned}$$

$$\begin{aligned}\vec{F}_{gnid} &= -\mu|\vec{N}|\frac{\dot{\vec{r}}}{|\dot{\vec{r}}|} \\ &= -\mu|mgcos\alpha|\begin{pmatrix}1\\0\end{pmatrix} \\ &= -\mu mgcos\alpha\begin{pmatrix}1\\0\end{pmatrix} \text{ (gennem definitionen af } \alpha \text{ og } g)\end{aligned}$$

Og den generaliseret kraft-led bestemmes:

$$\begin{aligned}Q_1 &= \vec{F}_{luft}\cdot\frac{\partial\vec{r}}{\partial R} + \vec{F}_{gnid}\cdot\frac{\partial\vec{r}}{\partial R} \\ \Leftrightarrow Q_1 &= -k\dot{R}\begin{pmatrix}\dot{R}\\0\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} - \mu mgcos\alpha\begin{pmatrix}1\\0\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} \\ &= -k\dot{R}^2 - \mu mgcos\alpha\end{aligned}$$

Og ved at tilføre den til Euler-Lagrange-ligningen får jeg:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial R} - Q_1 &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{R}}\right) \\ \Leftrightarrow mgsin\alpha - k\dot{R}^2 - \mu mgcos\alpha &= m\ddot{R} \\ \Leftrightarrow \ddot{R} &= gsin\alpha - \frac{k}{m}\dot{R}^2 - \mu gcos\alpha\end{aligned}$$

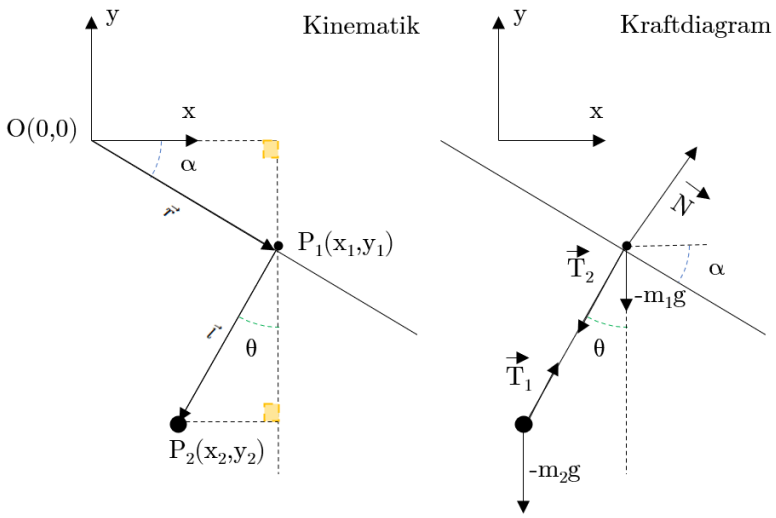
$$\boxed{\ddot{R} = gsin\alpha - \frac{k}{m}\dot{R}^2 - \mu gcos\alpha}$$

, som der findes en lang analytisk løsning for vist i bilag afsnit (10.1), hvor jeg har sat $k = \rho C_D A/2$, hvor:

- ρ er densiteten
- C_D er luftmodstandskoefficienten
- A er tværsnitsarealet af rytteren.

I forbindelse med mit program har dog jeg valgt at lade brugeren selv regne k ud for lettere implementation. Både den analytiske og numeriske løsning implementeres.

5.2 Det mere komplekse model



Jeg antager, at rebet er rigid dvs. den bukker sig ikke.

Rebet og vektor $\vec{r}$ med længde R danner en vinkel α med x-aksen, hvor y-aksen står modsatrettet tyngdekraftens bevægelsesretning.

Rytteren, sædet og den reelle snor har massemidpunkt ved P_2 og som punktmasse har massen m_2.

Punktmassen ved P_2 er bundet til punktmassen ved P_1 med en masseløs og stiv snor med længde $|\vec{l}| = L$, hvor $\vec{l}$ danner en vinkel θ med y-aksen.

Punktmassen ved P_1, som har massen m_1, skulle forestilles at være svævebanevognet, der holder snoren op.

Af hensyn til hvor store differentialligningerne bliver, hvis jeg inddrager ydre kræfter, antager jeg, at systemet ikke er påvirket af nogle ydre kræfter.

Følgende definitioner slås fast:

- $g = 9.81 \frac{\mathrm{m}}{\mathrm{s}^2}$
- $R, \theta \in \mathbb{R}$
- $\alpha \in \left\{ x \in \mathbb{R} : -\pi \leq x \leq 0 \right\}$

Systemet har to frihedsgrader og kan beskrives gennem to koordinater, R og θ. Som før, antages x_1, y_1, x_2, y_2, R og θs afhængighed af tiden, t, implicit.

Gennem triogenometriske overvejelser, kan positionerne for punktmasserne udtrykket af R og θ findes, som så kan differentieres for at finde hastighederne:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} -cos\alpha R \\ sin\alpha R \end{pmatrix} \qquad \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} -cos\alpha R + sin\theta L \\ sin\alpha R - cos\theta L \end{pmatrix}$$
$$\begin{pmatrix} \dot{x}_1 \\ \dot{y}_1 \end{pmatrix} = \begin{pmatrix} -cos\alpha \dot{R} \\ sin\alpha \dot{R} \end{pmatrix} \qquad \begin{pmatrix} \dot{x}_2 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} -cos\alpha \dot{R} + cos\theta\dot{\theta} L \\ sin\alpha \dot{R} + sin\theta\dot{\theta} L \end{pmatrix}$$

Den kinetiske og potentielle energi af de enkelte punktmasser findes:

$$\begin{aligned} U_1 &= m_1 g h_1 \\ &= m_1 g sin\alpha R \ (sin\alpha \text{ er negativt}) \\ T_1 &= \frac{1}{2} m_1 v_1^2 \\ &= \frac{1}{2} m_1 \left(\dot{x}_1^2 + \dot{y}_1^2 \right) \\ &= \frac{1}{2} m_1 \left((-cos\alpha \dot{R})^2 + (sin\alpha \dot{R})^2 \right) \\ &= \frac{1}{2} m_1 \left(cos^2\alpha \dot{R}^2 + sin^2\alpha R^2 \right) \\ &= \frac{1}{2} m_1 \left(\dot{R}^2 \left(sin^2\alpha + cos^2\alpha \right) \right) \\ &= \frac{1}{2} m_1 \dot{R}^2 \end{aligned}$$

, hvor $sin^2\alpha + cos^2\alpha = 1$ er en trigonometrisk identitet.[15]

$$\begin{aligned} U_2 &= m_2 g h_2 \\ &= m_2 g (sin\alpha R - cos\theta L) \\ &= m_2 g sin\alpha R - m_2 g cos\theta L \end{aligned}$$

, som man kan konstatere følgende for:

- Hvis θ holdes konstant, går U_2 mod $-\infty$, når tiden går mod $+\infty$. Det er i overenstemmelse med, at den potentielle energi bliver mindre, når P_2 går tættere mod jorden.
- Hvis α holdes konstant, er U_2 størst, når $\theta = \pi + 2\pi n$, og mindst, når $\theta = 2\pi n$, hvor n er et helt tal. Det er i overenstemmelse med et typisk penduls potentielle energi.

[15] University of Liverpool: »Trigonometric Identities«

- Hvor man vælger referencepunktet skal være, betyder ikke noget i sammenhæng med Euler-Lagrange-ligningen (2), da konstantleddene alligevel vil blive differentieret og forsvinde. Det skal bare være et punkt, pendulet kan nå, og det kan det ved en bestemt t og vinkel θ.

$$
\begin{aligned}
T_2 &= \frac{1}{2}m_2 v_2^2 \\
&= \frac{1}{2}m_2\left(\dot{x}_2^2 + \dot{y}_2^2\right) \\
&= \frac{1}{2}m_2\left(\left(-cos\alpha\dot{R} + cos\theta\dot{\theta}L\right)^2 + \left(sin\alpha\dot{R} + sin\theta\dot{\theta}L\right)^2\right) \\
&= \frac{1}{2}m_2\left(cos^2\alpha\dot{R}^2 + cos^2\theta\dot{\theta}^2L^2 - 2\dot{R}\dot{\theta}Lcos\alpha cos\theta + sin^2\alpha\dot{R}^2 + sin^2\theta\dot{\theta}^2L^2 + 2\dot{R}\dot{\theta}Lsin\alpha sin\theta\right) \\
&= \frac{1}{2}m_2\left(\dot{R}^2\left(sin^2\alpha + cos^2\alpha\right) + \dot{\theta}^2L^2\left(sin^2\theta + cos^2\theta\right) - 2\dot{R}\dot{\theta}L\left(cos\alpha cos\theta - sin\alpha sin\theta\right)\right)
\end{aligned}
$$

, hvor, da $sin^2\alpha + cos^2\alpha = 1$ og $cos\alpha cos\theta - sin\alpha sin\theta = cos(\alpha + \theta)$ er triogenometriske identiteter[16], kan ligningen ovenfor reduceres til følgende:

$$
\begin{aligned}
\Leftrightarrow T_2 &= \frac{1}{2}m_2\left(\dot{R}^2\left(sin^2\alpha + cos^2\alpha\right) + \dot{\theta}^2L^2\left(sin^2\theta + cos^2\theta\right) - 2\dot{R}\dot{\theta}L\left(cos\alpha cos\theta - sin\alpha sin\theta\right)\right) \\
&= \frac{1}{2}m_2\left(\dot{R}^2 + \dot{\theta}^2L^2 - 2\dot{R}\dot{\theta}Lcos(\alpha + \theta)\right) \\
&= \frac{1}{2}m_2\dot{R}^2 + \frac{1}{2}m_2\dot{\theta}^2L^2 - m_2\dot{R}\dot{\theta}Lcos(\alpha + \theta)
\end{aligned}
$$

Lagrange-funktionen (1) kan nu findes på følgende måde:

$$
\begin{aligned}
\mathcal{L} &= T_1 + T_2 - \left(U_1 + U_2\right) \\
&= \frac{1}{2}m_1\dot{R}^2 - \frac{1}{2}m_2\dot{R}^2 + \frac{1}{2}m_2\dot{\theta}^2L^2 - m_2\dot{R}\dot{\theta}Lcos(\alpha + \theta) - \left(m_1gsin\alpha R + m_2gsin\alpha R - m_2gcos\theta L\right) \\
&= \frac{1}{2}\dot{R}^2\left(m_1 + m_2\right) + \frac{1}{2}m_2\dot{\theta}^2L^2 - m_2\dot{R}\dot{\theta}Lcos(\alpha + \theta) - \left(gsin\alpha R\left(m_1 + m_2\right) - m_2gcos\theta L\right) \\
&= \frac{1}{2}\dot{R}^2\left(m_1 + m_2\right) + \frac{1}{2}m_2\dot{\theta}^2L^2 - m_2\dot{R}\dot{\theta}Lcos(\alpha + \theta) - gsin\alpha R\left(m_1 + m_2\right) + m_2gcos\theta L
\end{aligned}
$$

[16] University of Liverpool: »Trigonometric Identities«

$$\boxed{\mathcal{L} = \frac{1}{2}\dot{R}^2\left(m_1+m_2\right) + \frac{1}{2}m_2\dot{\theta}^2L^2 - m_2\dot{R}\dot{\theta}Lcos(\alpha+\theta) - gsin\alpha R\left(m_1+m_2\right) + m_2gcos\theta L}$$

Jeg får to Euler-Lagrange-ligninger (2) af de to koordinater, R og θ:

$$\frac{\partial\mathcal{L}}{\partial R} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{R}}\right) + Q_1$$
$$\frac{\partial\mathcal{L}}{\partial\theta} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{\theta}}\right) + Q_2$$

, hvor:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial R} &= -gsin\alpha\left(m_1+m_2\right)\\ \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{R}}\right) &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\dot{R}\left(m_1+m_2\right) - m_2\dot{\theta}Lcos(\alpha+\theta)\right)\\ &= \ddot{R}\left(m_1+m_2\right) - m_2\ddot{\theta}Lcos(\alpha+\theta) + m_2\dot{\theta}^2Lsin(\alpha+\theta)\end{aligned}$$

, hvor jeg anvender produktreglen, $(f\cdot g)' = f'g + fg'$ og indre-ydre-funktions regel, $(f(g(x)))' = g'f(g(x))$.

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\theta} &= m_2\dot{R}\dot{\theta}Lsin(\alpha+\theta) - m_2gsin\theta L\\ \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial\mathcal{L}}{\partial\dot{\theta}}\right) &= \frac{\mathrm{d}}{\mathrm{d}t}\left(m_2\dot{\theta}L^2 - m_2\dot{R}Lcos(\alpha+\theta)\right)\\ &= m_2\ddot{\theta}L^2 - m_2\ddot{R}Lcos(\alpha+\theta) + m_2\dot{R}\dot{\theta}Lsin(\alpha+\theta)\end{aligned}$$

, hvor jeg igen anvender produktreglen og indre-ydre-funktions regel.

Euler-Lagrange ligningerne kan så omskrives til:

$$\begin{aligned}-gsin\alpha\left(m_1+m_2\right) &= \ddot{R}\left(m_1+m_2\right) - m_2\ddot{\theta}Lcos(\alpha+\theta) + m_2\dot{\theta}^2Lsin(\alpha+\theta)\\ &\Leftrightarrow 0 = \ddot{R}\left(m_1+m_2\right) - m_2\ddot{\theta}Lcos(\alpha+\theta) + m_2\dot{\theta}^2Lsin(\alpha+\theta) + gsin\alpha\left(m_1+m_2\right)\\ &\Leftrightarrow 0 = \ddot{R} + gsin\alpha + \frac{1}{m_1+m_2}\cdot\left(m_2\dot{\theta}^2Lsin(\alpha+\theta) - m_2\ddot{\theta}Lcos(\alpha+\theta)\right)\end{aligned}$$

og:

$$\begin{aligned}m_2\dot{R}\dot{\theta}Lsin(\alpha+\theta) - m_2gsin\theta L &= m_2\ddot{\theta}L^2 - m_2\ddot{R}Lcos(\alpha+\theta) + m_2\dot{R}\dot{\theta}Lsin(\alpha+\theta)\\ &\Leftrightarrow 0 = m_2\ddot{\theta}L^2 - m_2\ddot{R}Lcos(\alpha+\theta) + m_2gsin\theta L\\ &\Leftrightarrow 0 = \ddot{\theta} + \frac{g}{L}sin\theta - \frac{1}{L}\ddot{R}cos(\alpha+\theta)\end{aligned}$$

Jeg vil nu finde to differentialligninger, hvor $\ddot{R}$ og $\ddot{\theta}$ står hver for sig. Lignignerne ovenfor kan omskrives til:

$$0 = a \cdot \ddot{R} + b \cdot \ddot{\theta} + c$$
$$0 = \ddot{\theta} + d \cdot \ddot{R} + e$$

, hvor:

- $a = \Big(m_1 + m_2\Big)$
- $b = -m_2 L cos(\alpha + \theta)$
- $c = m_2 \dot{\theta}^2 L sin(\alpha + \theta) + g sin\alpha \Big(m_1 + m_2\Big)$
- $d = -\frac{1}{L} cos(\alpha + \theta)$
- $e = \frac{g}{L} sin\theta$

Jeg nøjes med at bare bruge et CAS-værktøj til at løse ligningerne ovenfor, og får:

$$\boxed{\ddot{R} = \frac{be - c}{a - bd}} \qquad \boxed{\ddot{\theta} = \frac{ae - cd}{a - bd}}$$

5.3 Eksempel på anvendelsen af Eulers metode

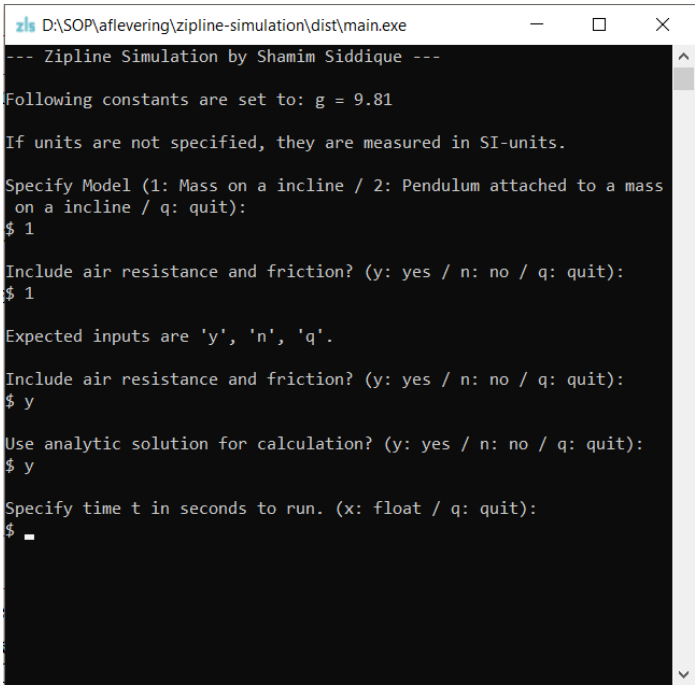
Disse to udtryk kan jeg bruge til simulere modellen gennem Eulers numeriske metode, som foregår på følgende måde:

1. Startværdierne for R, $\dot{R}$, θ og $\dot{\theta}$ samt m_1, m_2, α, L og dt bestemmes af brugeren.
2. Jeg bruger ligningerne ovenfor til at beregne $\ddot{R}_n$ og $\ddot{\theta}_n$.
3. $\dot{R}$ og $\dot{\theta}$ approksimeres gennem Eulers metode.
$$\dot{R}_n = \dot{R}_{n-1} + \ddot{R_n} \cdot dt$$
$$\dot{\theta}_n = \dot{\theta}_{n-1} + \ddot{\theta_n} \cdot dt$$
4. R og θ approksimeres gennem Eulers metode.
$$R_n = R_{n-1} + \dot{R}_n \cdot dt$$
$$\theta_n = \theta_{n-1} + \dot{\theta}_n \cdot dt$$
5. Jeg starter forfra fra trin 2 så mange gange, som der er brug for.

6 Beskrivelse af programmet

6.1 Funktionsbeskrivelse

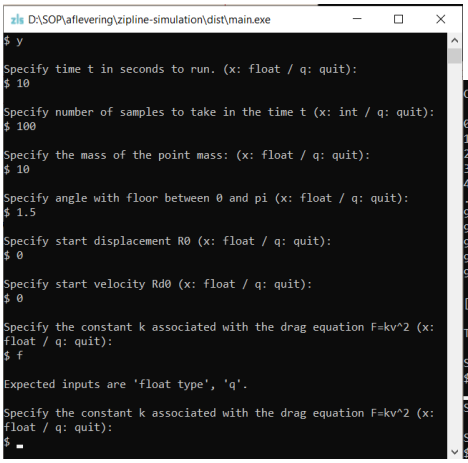
Brugeren interagerer med programmet gennem terminalen. Når man starter programmet op, vil en terminal-vindue åbne op, og man får lov til at vælge mellem forskellige modeller og antagelser.

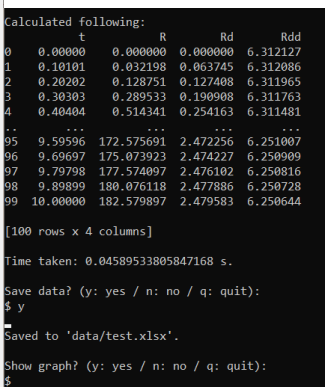


Man har fire valgmuligheder i alt ift. modeller og antagelser:

- Simple model med ingen ydre kræfter (og analytisk løsning).
- Simple model med ydre kræfter og analytisk løsning
- Simple model med ydre kræfter og numerisk løsning.
- Mere kompleks model og numerisk løsning.

Når man har lavet sine valgmuligheder omkring modeller og antagelser føres man videre til parameter-stadiet, hvor man angiver sine parametre. Herefter får man lov til at gemme sin data som excel-fil, hvorefter der vises tiden taget for at køre while-loopen til at beregne de forskellige værdier og gemme dem. En »d«-suffix bruges til at angive, at leddet foran d'et er differentieret en gang.

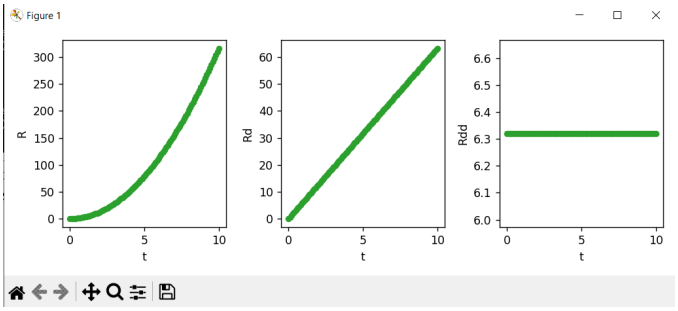




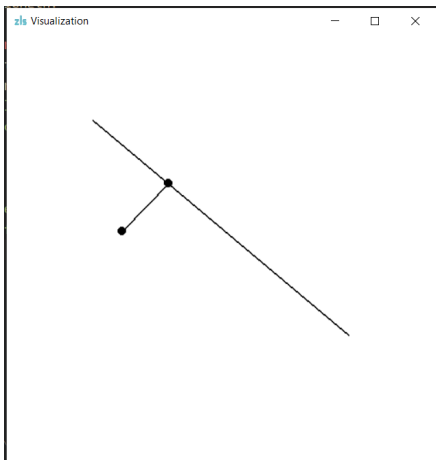
Excel-filerne er gemt under en mappe, 'data', relativ til programmets nuværende mappe. Mappen laves igen, hvis den ikke allerede eksisterer, og filerne er navngivet som 'test.xlsx', 'test_1.xlsx', 'test_2.xlsx' m. v.

	A	B	C	D	E
1	**t**	**R**	**Rd**	**Rdd**	
2	0	0	0	6,319011	
3	0,10101	0,032236	0,020184	6,31901	
4	0,20202	0,128941	0,040365	6,319009	
5	0,30303	0,290101	0,060541	6,319007	
6	0,40404	0,515696	0,080709	6,319004	
7	0,505051	0,805698	0,100867	6,319	
8	0,606061	1,160068	0,121012	6,318996	
9	0,707071	1,57876	0,141142	6,318991	
10	0,808081	2,061723	0,161253	6,318985	
11	0,909091	2,608892	0,181343	6,318978	

Herefter får man lov til at visualisere dataen gennem nogle grafer (altså en ja/nej spørgsmål stilles om det):



På samme tid med, at man også får lov til at visualisere dataen gennem en animation:



6.2 Dokumentation

6.2.1 Moduler

Standard library moduler	**Funktion**
math	Bruge adskillige matematiske funktioner.
multiprocessing	Køre numpy-graferne og pygame-animations-vinduet på samme tid.
time	Måle køretid.
os	Arbejde med filer udenfor programmet.
sys	Slukke programmet.
Eksterne moduler	
numpy	Tegne grafer
pandas	Opbevare data.
pygame	Animere data.
openpyxl	Brugt af pandas til at gemme dataen til en excel fil.
pyinstaller	Kompilere programmet.
Egne moduler	
inputparser	Interegere med brugeren og lede efter bestemte input typer.
data	Abstraktion af pandas Dataframes.
graph	Abstraktion af numpy grafer.
visualizer	Abstraktion af pygame.

6.2.2 main

Inkluderer funktionerne main, model1_wo_external_forces, model1_w_external_forces og model2. Disse funktioner bruger inputparser til at interegere med brugeren og lede efter model, antagelser, parametre, og andre svarer til valgmuligheder, og bruger eulers metod og DataHolder fra data modulen til at hhv. udregne og opbevare dataen. Herefter spørger den ind om brugeren vil gemme dataen som excel fil, grafere dataen og/eller visualisere dataen gennem en animation.

6.2.3 submodules.inputparser

Denne modul indeholder klassen InputParser, der bruges af main til at interegere med brugeren. De eksterne funktioner af klassen er start, get_input og get_type_input. Det førstnævnte printer en banner med navn ud, den anden printer en spørgsmål ud og leder efter inputs specificeret i possible_inputs, og

det sidstnævnte leder efter besteme input typer og kan tage imod en interval for at tjekke om inputtet ligger i intervallet. Klassen holder en slags »while«-loop kørende baseret på recursion ved at bruge returnere en funktion fra den samme funktion, hvis der er fejl i inputtet.

6.2.4 submodules.data

Indeholder DataHolder klassen, som abstraherer pandas dataframes med lettere syntaks. Indeholder også euler funktionen som følgende:

```
def euler(y0, h, f_val):
    return y0 + h * f_val
```

6.2.5 submodules.graph

Indeholder three_axes og six_axes, der hhv. tager imod Dataframes for det simple og mere kompleks model. Funktionerne abstraherer producering af numpy grafer.

6.2.6 submodules.visualizer

Inderholder animate funktionen, der visualiserer dataen gennem en animation på skærmen vha. pygame.

6.3 Test af programmet

Ved at sætte $t = 10$ og samples til 10000, hvor modellerne får parametre, der er nogenlunde ækvivalent med hinanden, får jeg følgende:

Indeks	Model	Køretid
1	Simple model med ingen ydre kræfter (og analytisk løsning).	4.37 s
2	Simple model med ydre kræfter og analytisk løsning	4.44 s
3	Simple model med ydre kræfter og numerisk løsning.	4.89 s
4	Mere kompleks model og numerisk løsning.	11.10 s

7 Diskussion af hvad der kunne være en passende model

Det viser sig, at de analytiske løsninger er hurtigere end de numeriske løsning. Det kan forklares af, at der muligivs udføres flere led af beregninger, og forskellen, der ses mellem model 2 og 3, opstår. Alligevel er det ikke en stor forskel og vil man ikke have det performance og realisme, som i dette tilfælde ikke er omvendt proportionale, for at undgå at skrive formlen ind, kan man benytte sig af numeriske metoder.

Model 4 med væsentlige mange flere led ift. de andre modeller viser sig at have det største køretid. Den bruger cirka dobbelt så meget tid ift. de andre modeller. Her er performance og realisme faktisk omvendt proportionale.

Hvis man vil inddrage model 4 i sit spil, kunne man dog beregne en bane på forhånd med parametre bestemt på forhånd. Hvis man for eksempel ved playerens masse og svævevognets masse og sætter nogle af de andre parametere til konstanter, kunne man få en model, der kører ligeså hurtigt som de andre modeller. Her udgør aflæsningen af dataen »bottleneck«-en, da man ikke længere skal udregne banen, især hvis sit datasæt er stor.

8 Konklusion

Lagranges formalisme anvender Hamiltons princip til at udlede ligningerne for bevægelse i et system. Hamiltons princip siger, at den aktuelle bane en partikel tager, er hvor virkningsintergralet er stationær.

Gennem Lagranges formalisme jeg har fundet frem til nogle ligninger for bevægelse i svævebaner. De har taget hensyn til eksterne kræfter som luftmodstandskraft og gnidningskraft samt rytterens egen bevægelse.

Gennem et række python moduler har jeg lavet et program, gennem hvilket testresultaterne siger, at de analytiske løsninger er hurtigere end de numeriske løsninger, og inddragelse af luftmodstandskraft og gnidningskraften i svævebaner som skrå plan med en genstand ikke ændrer køretiden særlig meget.

Alt i alt, er der et række udfordringer i at modellere en realistisk bevægelse i en svævebane. Udover at man skal skrive store ligninger ind i programmet, hvis man vil øve realismen, når man til et punkt, hvor differentialligningerne ikke længere kan løses analytisk og kræver en del mere køretid at blive løst numerisk. Kan man beregne banen på forhåndet, er det dog muligt at implementere mere realistiske bevægelser.

9 Kildeliste

Britannica, T. Redaktører of Encyclopaedia. "Lagrangian function." Encyclopedia Britannica, https://www.britannica.com/science/Lagrangian-function (lokaliseret d. 17 december 2022).

Pygame, 2017: »Recipe Multiprocessing«, https://github.com/pyinstaller/pyinstaller/wiki/Recipe-Multiprocessing (lokaliseret d. 20 december 2022).

J. Kim Vandiver, 2011a: R9. Generalized Forces, MIT OpenCourseWare https://youtu.be/QadsG49DY3M(lokaliseret d. 17 december 2022).

J. Kim Vandiver, 2011b: 16. Kinematic Approach to Finding Generalized Forces, MIT OpenCourseWare, 28:46-34:09 https://youtu.be/lFedznDnPZc?t=1726 (lokaliseret d. 17 december 2022).

J. R. Taylor: Classical Mechanics, University Science Books: California.

Stackoverflow, 2017: »Create file but if name exists add number«, https://stackoverflow.com/a/57896232 (lokaliseret d. 20 december 2022)

Stackoverflow, 2018: »How to disable welcome message when importing pygame«, https://stackoverflow.com/a/55769463 (lokaliseret d. 20 december 2022)

Matplotlib: »Introductory«, https://matplotlib.org/stable/tutorials/introductory/index.html (lokaliseret 20/12/2022)

NumPy: »NumPy Reference«, https://numpy.org/doc/stable/reference/index.html (lokaliseret 20/12/2022)

Pygame: »pygame documentation«, https://www.pygame.org/docs/ (lokaliseret 20/12/2022)

University of Liverpool: Trigonometric Identities, https://www.liverpool.ac.uk/~maryrees/homepagemath191/t (lokaliseret d. 17 december 2022).

Wikipedia bidragsydere, "Generalized forces," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Generalized_forces&oldid=798161881 (lokaliseret 17/12/2022)

Wikipedia bidragsydere, "Hamilton's principle," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Hamilton%27s_principle&oldid=1117662115 (lokaliseret d. 17 december 2022).

Wikipedia bidragsydere, "Joseph-Louis Lagrange," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Joseph-Louis_Lagrange&oldid=1121816812

(lokaliseret d. 17 december 2022).

Wikipedia bidragsydere, "Leonhard Euler," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Leonhard_Euler&oldid=1127824460 (lokaliseret d. 17 december 2022).

Wikipedia bidragsydere, "Lille vinkel", Wikipedia, Den frie encyklopædi., https://da.wikipedia.org/w/index.php?title=Lille_vinkel&oldid=11105739 (lokaliseret d. 17 de- cember 2022)

10 Bilag

10.1 Analytiske løsning til bevægelsesligningen i den simple model

10.1.1 Udtryk for $\ddot{R}$

$$\ddot{R} = g(sin\alpha - \mu cos\alpha) - \frac{\rho C_D A}{2m}\dot{R}^2$$

10.1.2 Udtryk for $\dot{R}$

$$\dot{R} = \frac{\sqrt{k_1} \cdot tanh\left(\dot{R}_0\sqrt{k_1}\sqrt{k_2} + t\sqrt{k_1}\sqrt{k_2}\right)}{\sqrt{k_2}}, k_1 = g(sin\alpha - \mu cos\alpha), k_2 = \frac{\rho C_D A}{2m}$$

10.1.3 Udtryk for R

$$R = R_0 + \frac{ln\left(cosh\left(\dot{R}_0\sqrt{k_1}\sqrt{k_2} + t\sqrt{k_1}\sqrt{k_2}\right)\right)}{k_2}, k_1 = g(sin\alpha - \mu cos\alpha), k_2 = \frac{\rho C_D A}{2m}$$

10.2 Kode

```
"""
main.py

External Dependencies:
- numpy
- pandas
- matplotlib
- pygame
- openpyxl (not required)
- pyinstaller (required for compiling)
"""
import math
import multiprocessing
from time import time

from submodules import graph, visualizer
from submodules.data import DataHolder, euler
from submodules.inputparser import InputParser

# Constants:
g = 9.81


def main():
    ip = InputParser()
    ip.start()
    print("\nFollowing constants are set to: g = 9.81")
    print("\nIf units are not specified, they are measured in SI-units.")
    model = ip.get_input(
        question="Specify Model",
        input_description=(
            "1: Mass on a incline",
            "2: Pendulum attached to a mass on a incline",
        ),
        possible_inputs=("1", "2"),
    )
    if model == "1":
        answer = ip.get_input(
            question="Include air resistance and friction?",
            input_description=(
                "y: yes",
                "n: no",
            ),
            possible_inputs=("y", "n"),
        )
        if answer == "y":
            model1_w_external_forces(ip)
        else:
            model1_wo_external_forces(ip)
    else:
        model2(ip)


def model1_wo_external_forces(ip: InputParser):
    t = ip.get_type_input("Specify time t in seconds to run.", float)
    samples = ip.get_type_input("Specify number of samples to take in the time t", int)
    a = ip.get_type_input(
        "Specify angle with floor between 0 and pi", float, interval=(0, math.pi)
    )
    R0 = ip.get_type_input("Specify start displacement R0", float)
    Rd0 = ip.get_type_input("Specify start velocity Rd0", float)

    k = g * math.sin(a)

    dh = DataHolder(t, samples)
    dh.set_val(0, "R", R0)
    dh.set_val(0, "Rd", Rd0)
```

```
    dh.set_val(0, "Rdd", k)

    i = 1
    l = dh.length

    t0 = time()
    while i < l:
        t = dh.get_val(i, "t")
        dh.set_val(i, "Rdd", k)
        dh.set_val(i, "Rd", k * t + Rd0)
        dh.set_val(i, "R", 1 / 2 * k * t**2 + Rd0 * t + R0)
        i += 1
    t1 = time() - t0
    print(f"\nCalculated following:")
    print(dh.data)
    print(f"\nTime taken: {t1} s.")
    answer = ip.get_input(
        question="Save data?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    answer = ip.get_input(
        question="Show graph?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    p1 = multiprocessing.Process(target=graph.three_axes, args=(dh.data,))
    p2 = multiprocessing.Process(target=visualizer.animate, args=(dh, samples / t, a))

    if answer == "y":
        p1.start()
    answer = ip.get_input(
        question="Show animation?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    if answer == "y":
        p2.start()


def model1_w_external_forces(ip: InputParser):
    answer = ip.get_input(
        question="Use analytic solution for calculation?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    use_analytic_solution = answer == "y"
    t = ip.get_type_input("Specify time t in seconds to run.", float)
    samples = ip.get_type_input("Specify number of samples to take in the time t", int)
    m = ip.get_type_input("Specify the mass of the point mass:", float)
    a = ip.get_type_input(
        "Specify angle with floor between 0 and pi", float, interval=(0, math.pi)
    )
    R0 = ip.get_type_input("Specify start displacement R0", float)
    Rd0 = ip.get_type_input("Specify start velocity Rd0", float)
```

```
    k = ip.get_type_input(
        "Specify the constant k associated with the drag equation F=kv^2", float
    )
    u = ip.get_type_input("Specify the coefficient u", float)

    if use_analytic_solution:
        k1 = g * math.sin(a) - u * math.cos(a)
        k2 = k / m
        k1sqr = math.sqrt(k1)
        k2sqr = math.sqrt(k2)
        k1k2sqrt = k1sqr * k2sqr

        dh = DataHolder(t, samples)
        dh.set_val(0, "R", R0)
        dh.set_val(0, "Rd", Rd0)
        dh.set_val(0, "Rdd", k1 - k2 * Rd0**2)

        i = 1
        l = dh.length

        t0 = time()
        while i < l:
            t = dh.get_val(i, "t")
            dh.set_val(i, "R", R0 + math.log(math.cosh((Rd0 + t) * k1k2sqrt)) / k2)
            Rd = k1k2sqrt * math.tanh((Rd0 + t) * k1k2sqrt) / k2sqr
            dh.set_val(i, "Rd", Rd)
            dh.set_val(i, "Rdd", k1 - k2 * Rd**2)

            i += 1
        t1 = time() - t0
    else:
        k1 = g * math.sin(a) - u * math.cos(a)
        k2 = k / m

        dh = DataHolder(t, samples)
        dh.set_val(0, "R", R0)
        dh.set_val(0, "Rd", Rd0)
        dh.set_val(0, "Rdd", k1 - k2 * Rd0**2)

        i = 1
        l = dh.length

        dt = dh.get_val(1, "t") - dh.get_val(0, "t")
        Rdd_function = lambda Rd: k1 - k2 * Rd**2

        t0 = time()
        while i < l:
            Rdnm1 = dh.get_val(i - 1, "Rd")
            Rnm1 = dh.get_val(i - 1, "R")

            Rdd = Rdd_function(Rdnm1)
            Rd = euler(Rdnm1, dt, f_val=Rdd)
            R = euler(Rnm1, dt, f_val=Rdd)

            dh.set_val(i, "R", R)
            dh.set_val(i, "Rd", Rd)
            dh.set_val(i, "Rdd", Rdd)

            i += 1
        t1 = time() - t0

    print(f"\nCalculated following:")
    print(dh.data)
    print(f"\nTime taken: {t1} s.")
    answer = ip.get_input(
        question="Save data?",
        input_description=(
```

```
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    if answer == "y":
        print(f"\nSaved to '{dh.save()}'.")

    answer = ip.get_input(
        question="Show graph?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    p1 = multiprocessing.Process(target=graph.three_axes, args=(dh.data,))
    p2 = multiprocessing.Process(target=visualizer.animate, args=(dh, samples / t, a))

    if answer == "y":
        p1.start()
    answer = ip.get_input(
        question="Show animation?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    if answer == "y":
        p2.start()


def model2(ip: InputParser):
    t = ip.get_type_input("Specify time t in seconds to run.", float)
    samples = ip.get_type_input("Specify number of samples to take in the time t", int)
    m1 = ip.get_type_input("Specify mass of the zipline clip / trolley", float)
    m2 = ip.get_type_input("Specify mass of the zipline rider", float)
    l = ip.get_type_input(
        "Specify the length of the string between the trolley and the rider", float
    )
    alp = ip.get_type_input(
        "Specify angle with x-axis between -pi and 0", float, interval=(-math.pi, 0)
    )
    R0 = ip.get_type_input("Specify start displacement R0", float)
    Rd0 = ip.get_type_input("Specify start velocity Rd0", float)
    th0 = ip.get_type_input("Specify start angle of the rider th0", float)
    thd0 = ip.get_type_input("Specify start angular velocity of the rider thd0", float)

    a = m1 + m2
    b = lambda th: -m2 * l * math.cos(alp + th)
    c = lambda thd, th: m2 * thd**2 * l * math.sin(alp + th) + g * math.sin(alp) * (
        m1 + m2
    )
    d = lambda th: -1 / l * math.cos(alp + th)
    e = lambda th: g / l * math.sin(th)

    Rdd_function = lambda th, thd: (b(th) * e(th) - c(thd, th)) / (a - b(th) * d(th))
    thdd_function = lambda th, thd: (a * e(th) - c(thd, th) * d(th)) / (
        a - b(th) * d(th)
    )

    dh = DataHolder(t, samples)
    dh.set_val(0, "R", R0)
    dh.set_val(0, "Rd", Rd0)
    dh.set_val(0, "Rdd", Rdd_function(th0, thd0))
    dh.set_val(0, "th", th0)
```

```
    dh.set_val(0, "thd", thd0)
    dh.set_val(0, "thdd", thdd_function(th0, thd0))

    i = 1
    l = dh.length

    dt = dh.get_val(1, "t") - dh.get_val(0, "t")
    t0 = time()

    while i < l:
        Rdnm1 = dh.get_val(i - 1, "Rd")
        Rnm1 = dh.get_val(i - 1, "R")
        thdnm1 = dh.get_val(i - 1, "thd")
        thnm1 = dh.get_val(i - 1, "th")

        thdd = thdd_function(thnm1, thdnm1)
        thd = euler(thdnm1, dt, f_val=thdd)
        th = euler(thnm1, dt, f_val=thdd)

        Rdd = Rdd_function(thnm1, thdnm1)
        Rd = euler(Rdnm1, dt, f_val=Rdd)
        R = euler(Rnm1, dt, f_val=Rdd)

        dh.set_val(i, "R", R)
        dh.set_val(i, "Rd", Rd)
        dh.set_val(i, "Rdd", Rdd)

        dh.set_val(i, "th", th)
        dh.set_val(i, "thd", thd)
        dh.set_val(i, "thdd", thdd)

        i += 1
    t1 = time() - t0

    print(f"\nCalculated following:")
    print(dh.data)
    print(f"\nTime taken: {t1} s.")
    answer = ip.get_input(
        question="Save data?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    if answer == "y":
        print(f"\nSaved to '{dh.save()}'.")

    answer = ip.get_input(
        question="Show graph?",
        input_description=(
            "y: yes",
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    p1 = multiprocessing.Process(target=graph.six_axes, args=(dh.data,))
    p2 = multiprocessing.Process(
        target=visualizer.animate, args=(dh, samples / t, alp, l)
    )

    if answer == "y":
        p1.start()
    answer = ip.get_input(
        question="Show animation?",
        input_description=(
            "y: yes",
```

```
            "n: no",
        ),
        possible_inputs=("y", "n"),
    )
    if answer == "y":
        p2.start()


if __name__ == "__main__":
    # https://github.com/pyinstaller/pyinstaller/wiki/Recipe-Multiprocessing
    multiprocessing.freeze_support()
    main()
```

```python
"""
submodules/data.py
"""
import os

import numpy as np
import pandas as pd


def euler(y0, h, f_val):
    return y0 + h * f_val


# https://stackoverflow.com/questions/13852700/create-file-but-if-name-exists-add-number
def uniquify(path):
    filename, extension = os.path.splitext(path)
    counter = 1

    while os.path.exists(path):
        path = filename + "_" + str(counter) + extension
        counter += 1

    return path


class DataHolder:
    def __init__(self, total_time_s: float, samples: int):
        self.data = pd.DataFrame({"t": np.linspace(0, total_time_s, samples)})

        if not os.path.exists("data"):
            os.makedirs("data")

    def set_val(self, i, column, val):
        rowIndex = self.data.index[i]
        self.data.loc[rowIndex, column] = val

    def get_val(self, i, column):
        return self.data.at[i, column]

    def save(self):
        path = uniquify("data/test.xlsx")
        self.data.to_excel(path, index=False)
        return path

    @property
    def length(self):
        return self.data.shape[0]
```

```python
"""
submodules/graph.py
"""
import matplotlib.pyplot as plt
from pandas import DataFrame


def three_axes(df: DataFrame):
    plt.rcParams["figure.autolayout"] = True  # better layout
    fig, ax = plt.subplots(ncols=3, figsize=(9, 6), dpi=100)
    df.plot(kind="scatter", x="t", y="R", ax=ax[0], color="tab:green")
    df.plot(kind="scatter", x="t", y="Rd", ax=ax[1], color="tab:green")
    df.plot(kind="scatter", x="t", y="Rdd", ax=ax[2], color="tab:green")
    plt.show()


def six_axes(df: DataFrame):
    plt.rcParams["figure.autolayout"] = True  # better layout
    fig, ax = plt.subplots(nrows=2, ncols=3, figsize=(9, 12), dpi=100)
    df.plot(kind="scatter", x="t", y="R", ax=ax[0, 0], color="tab:green")
    df.plot(kind="scatter", x="t", y="Rd", ax=ax[0, 1], color="tab:green")
    df.plot(kind="scatter", x="t", y="Rdd", ax=ax[0, 2], color="tab:green")
    df.plot(kind="scatter", x="t", y="th", ax=ax[1, 0], color="tab:blue")
    df.plot(kind="scatter", x="t", y="thd", ax=ax[1, 1], color="tab:blue")
    df.plot(kind="scatter", x="t", y="thdd", ax=ax[1, 2], color="tab:blue")
    plt.show()
```

```
"""
submodules/inputparser.py
"""
import sys


class InputParser:
    def __init__(self):
        pass

    def _question(self, question, input_description):
        print()
        print(f"{question} ", end="")

        print("(", end="")
        count = len(input_description)
        countm1 = count - 1

        for i, inp in enumerate(input_description):
            print(inp, end="")
            if i < count:
                print(" / ", end="")
            if i == countm1:
                print("q: quit", end="")

        print("):")
        print("$ ", end="")

    def _warn(self, possible_inputs):
        print(f"\nExpected inputs are", end="")
        count = len(possible_inputs)
        for i, p_inp in enumerate(possible_inputs):
            print(f" '{p_inp}'", end="")
            if i < count:
                print(f",", end="")
        print(" 'q'.")

    def start(self):
        print("--- Zipline Simulation by Shamim Siddique ---")

    def get_input(
        self, question: str, input_description: list[str], possible_inputs: list[str]
    ):
        self._question(question, input_description)

        inp = input().strip()
        if inp == "q":
            sys.exit()

        if inp not in possible_inputs:
            self._warn(possible_inputs)
            # recursive loop
            return self.get_input(question, input_description, possible_inputs)
        return inp

    def get_type_input(
        self,
        question: str,
        input_type: type,
        interval: tuple[float, float] | None = None,
    ):
        self._question(question, [f"x: {input_type.__name__}"])

        inp = input().strip()

        if inp == "q":
            sys.exit()
```

```
    try:
        val = input_type(inp)
        if interval is not None:
            if val < interval[0] or val > interval[1]:
                print()
                print(f"Value must lie between {interval[0]} and {interval[1]}.")
                return self.get_type_input(question, input_type, interval)
        return val
    except:
        self._warn([f"{input_type.__name__} type"])
        return self.get_type_input(question, input_type)
```

```
"""
submodules/visualizer.py
"""
import math
import os

import numpy as np

from submodules.data import DataHolder


def animate(
    dh: DataHolder,
    frames: int,
    alpha: float,
    L: float | None = None,
):
    # https://stackoverflow.com/questions/51464455/how-to-disable-welcome-message-when-importing-pygame/51470016#51470016
    os.environ["PYGAME_HIDE_SUPPORT_PROMPT"] = "hide"

    import pygame as pg

    pg.init()
    pg.display.set_caption("Visualization")
    pg.display.set_mode((500, 500), pg.SCALED | pg.RESIZABLE)

    R_min = dh.data["R"].iloc[0]
    R_max = dh.data["R"].iloc[-1]

    if L:
        th_start = dh.data["th"].iloc[0]
        th_end = dh.data["th"].iloc[-1]
        costh_start = math.cos(th_start)
        costh_end = math.cos(th_end)
        sinth_start = math.sin(th_start)
        sinth_end = math.sin(th_end)

    running = True
    clock = pg.time.Clock()

    i = 0
    radius = 5
    end_size = np.array(((300 - 2 * radius), (300 - 2 * radius)))
    cosalpha = math.cos(alpha)
    sinalpha = math.sin(alpha)

    if L:  # flip alpha if it's the more complex model per definition.
        sinalpha = -sinalpha

    R0 = dh.data["R"].iloc[0]

    if not L:
        x_min = cosalpha * R_min
        y_min = sinalpha * R_min
        x_max = cosalpha * R_max
        y_max = sinalpha * R_max
    else:
        x_min = cosalpha * R_min + min(sinth_start * L,0)
        y_min = sinalpha * R_min + min(costh_start * L,0)
        x_max = cosalpha * R_max + max(sinth_end * L,0)
        y_max = sinalpha * R_max + max(costh_end * L,0)

    x_diff = x_max - x_min
    y_diff = y_max - y_min

    first_pos = (100, 100) + end_size * (
```

```
        (cosalpha * R0 - x_min) / x_diff,
        (sinalpha * R0 - x_min) / y_diff,
    )
    R1 = dh.data["R"].iloc[-1]
    last_pos = (100, 100) + end_size * (
        (cosalpha * R1 - x_min) / x_diff,
        (sinalpha * R1 - x_min) / y_diff,
    )

    surface = pg.display.get_surface()
    while running:
        clock.tick(frames)

        R = dh.get_val(i, "R")

        pos1 = (100, 100) + end_size * (
            (cosalpha * R - x_min) / x_diff,
            (sinalpha * R - y_min) / y_diff,
        )
        if L:
            theta = dh.get_val(i, "th")
            costheta = math.cos(theta)
            sintheta = math.sin(theta)
            pos2 = pos1 + end_size * (
                (sintheta * L-x_min) / x_diff,
                (costheta * L-y_min) / y_diff,
            )
        i += 1
        if i == dh.length:
            i = 0
        for event in pg.event.get():
            if event.type == pg.QUIT:
                running = False
        surface.fill((255, 255, 255))
        pg.draw.circle(surface, (0, 0, 0), pos1, radius)
        if L:
            pg.draw.circle(surface, (0, 0, 0), pos2, radius)
            pg.draw.line(surface, (0, 0, 0), pos1, pos2, width=2)
        pg.draw.line(surface, (0, 0, 0), first_pos + (0, radius), last_pos+(0, radius),
width=2)
        pg.display.flip()
```