Blidens mekanik

Titel	Blidens mekanik
Udarbejdet af	Henrik Skov Midtiby
Vejleder	Peter Touborg
Institut	Institut for Fysik og Kemi Syddansk Universitet
Afleveringsdato	31. maj 2006

Forord

Dette projekt er udarbejdet som et bachelor projekt, og er rettet mod personer med et kendskab til klassisk mekanik. Jeg vil gerne takke min vejleder, Peter Touborg, for at have tid når det er nødvendigt, for altid at være så begejstret, og for at have givet en god introduktion til Hamilton–Jacobi metoden. En tak til Per Lyngs Hansen, der har introduceret mig til dimensionsanalysen, og som har givet konstruktiv feedback på nogle af mine ideer. Desuden vil jeg takke Ingo Nielsen for meget konstruktiv og brugbar kritik i korrekturlæsnings fasen.

Min interesse for blider startede, da jeg gik i 6. klasse. Vi havde lige fået et nummer af bladet Illustreret Videnskab ind af døren. I lige præcis dette nummer var der en fire siders artikel [Vemming(1996)] omkring bliden, og hvordan den var blevet brugt som belejrings våben i middelalderen. Efter at have læst artiklen var der kun et at gøre: at bygge sin egen blide! I løbet af eftermiddagen byggede jeg så en funktionel model af bliden i Lego; modellen kunne kaste små glaskugler ca. $2m$. Senere samme år byggede jeg en ny blide i træ.

Så gik der et par år, inden jeg prøvede igen. Jeg var nu nået til 10. klasse og havde bl.a. metalsløjd. Et af mine projekter var en blide. Resultatet var en maskine, der vha. $25kg$ ballast kunne kaste en vandballon omkring $20m$.

Vi når videre til en gang i det tredje år på HTX. Jeg har fået den ide at jeg gerne vil lave en computer simulering af en blide. Jeg går hurtigt fast i beregningsdelen, da jeg ikke kan opstille nogle brugbare ligninger for hvilke kræfter, de enkelte dele af bliden bliver udsat for i en given situation.

Jeg starter på fysik studiet på SDU, og efter basisåret bliver vi introduceret til den klassiske mekanik, herunder Lagrange og Hamilton formalismen. Endelig var jeg i stand til at opstille de nødvendige bevægelsesligninger og derefter lave simuleringer af bliden.

Odense 31. maj 2006

Henrik Skov Midtiby

Resume

De første udgaver af bliden kommer til Europa i det 6. århundrede, hvor de kommer til at spille en rolle som et frygtet belejringsvåben gennem de næste 1000 år.

En blides konstruktion beskrives og der gives en kort gennemgang af blidens historie og udvikling. De opnåede resultater i projektet gengives kort og præcist i konklusionen, der er medtaget herunder.

Konklusion

En simpel model af en blide med hængslet ballast er beskrevet. I modellen ses der bort fra friktion og elastiske kræfter.

Vha. dimensionsanalyse, er det undersøgt hvordan skudlængde og optimalt skudtidspunkt skalerer. Hvis alle dimensioner (længder) af bliden øges med en faktor γ:

- øges skudlængden med en faktor γ
- øges skudtiden med en faktor $\sqrt{\gamma}$

Der er fundet bevægelsesligninger for modellen vha. Lagrange formalismen.

Der er forsøgt at finde en analytisk løsning til bevægelsesligningerne vha. Hamilton–Jacobi metoden, det lykkes ikke, da det ikke er muligt at lave separation af variable på et bestemt udtryk. At Hamilton–Jacobi metoden fejler, udelukker ikke, at der findes en analytisk løsning til bevægelsesligningerne.

Til at sandsynliggøre at der ikke findes en analytisk løsning, benyttes et argument fra beregnelighedsteori. Resultatet er, at hvis der findes en analytisk løsning til blidens bevægelse, kan denne løsning reduceres til et kaotisk pendul, som sandsynligvis ikke har en analytisk løsning.

Der implementeres en numerisk løsning af de opstillede bevægelsesligninger. Den numeriske løsning er baseret på en fjerde ordens Runge–Kutta metode.

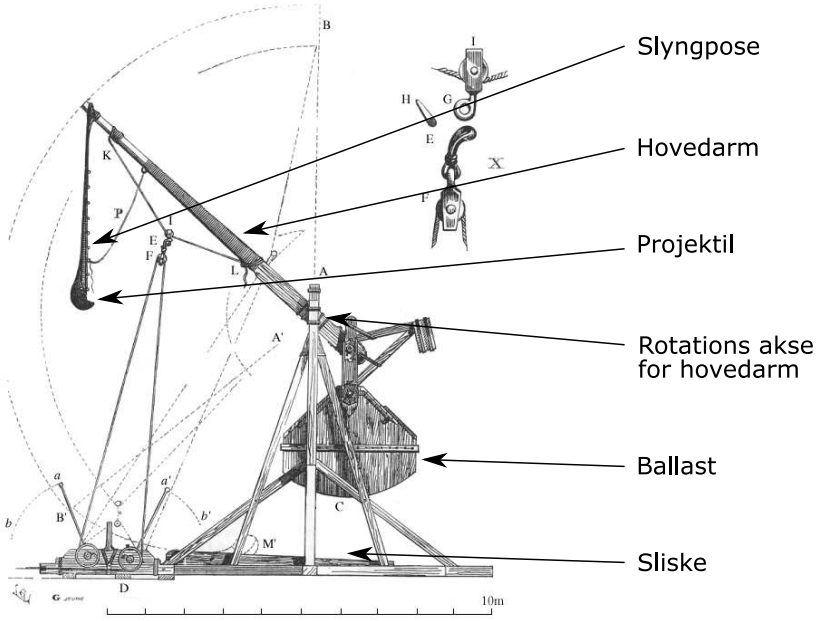
Rigtigheden af den implementerede løsning sandsynliggøres, ved at bestemme metodens orden ud fra de resultater som den numeriske løsning giver. Fejlen i den konstante energi konvergerer som $O(h^4)$, og fejlen på den maksimale skudlængde konvergerer som $O(h^2)$, hvilket i begge tilfælde er som forventet.

Til sidst er det undersøgt hvordan energien flyttes rundt i bliden under en affyring. Her findes det, at for en optimeret blide (med en masseløs hovedarm) er virkningsgraden 85%.

Indhold

Figurer

Tabeller



Figur 1: *Skitse af en moderne blide, med forklaringer til de i teksten omtalte dele. Illustration fra [Wikipedia(2006)].*

1 Introduktion

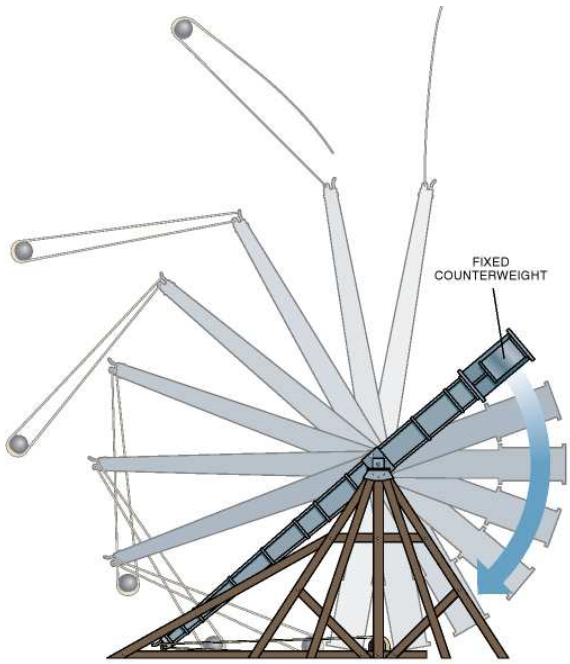
1.1 Hvad er en blide?

En blide er en gammeldags krigsmaskine. Bliden udmærker sig ved at være i stand til meget effektivt at overføre beliggenheds energi til bevægelses energi i et projektil. Dette blev benyttet til at slynge forskellige projektiler mod eller over en borgmur.

Den moderne blide består af en lang hovedarm, der roterer omkring en fast aksel. I den ene ende af hovedarmen hænger ballasten, der er en stor kasse, der kan fyldes med noget tungt. I den anden ende er der monteret en lang slyngpose, i hvilken projektilet ligger. Afstanden mellem hovedakslen og ballastens ophængningspunkt er, for en typisk blide, en fjerdedel af afstanden mellem hovedakslen og slyngposens fæstning. I hvile står hovedarmen lige op i vejret, så ballasten er nederst. Når bliden skal lades, vippes hovedarmen ned og holdes fast, hvorefter projektilet lægges i slyngposen. Når hovedarmen ikke længere holdes fast, vil den begynde at rotere og slyngposen følger med imens den accelereres i en rotationsbevægelse. På et tidspunkt (bestemt af vinklen mellem slyngposen og hovedarmen) frigives projektilet, der nu vil fortsætte sin bane mod målet.

1.2 Historie og udvikling

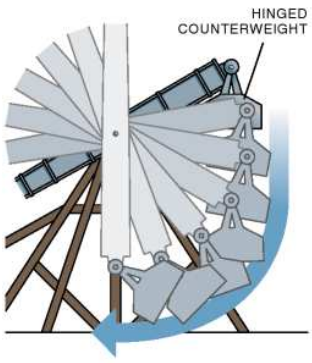
De første udgaver af en maskine med en virke-måde som blidens stammer fra Asien 300 til 500 år før vor tidsregning [Chevedden and Eigenbrod(1995)]. Den første udgave af bliden kommer til Europa i det 6. århundrede [Chevedden(2000)],



Figur 3: *Skitse af en blide i funktion. Det ses, at slyngposen frigiver projektilet på et (næsten) optimalt tidspunkt, da projektilet efter frigivelsen bevæger sig i en retning, der er ca. 45° i forhold til vandret. Bemærk hvordan projektilet bevæger sig med en meget højere hastighed end krogen for enden af hovedarmen. Illustration fra [Chevedden and Eigenbrod(1995)].*

der er tale om en såkaldt *traction trebuchet*, den drives af mennesker, der trækker i tove fastgjort til den korte ende af hovedarmen. I det 8. århundrede kommer der en hybrid-udgave af bliden, der bliver drevet af en kombination af folk, der trækker i snore og en ballast, der er fastmonteret på hovedarmen.

Sidst i det 12. århundrede kommer en ny udgave af bliden frem. Denne udgave benytter kun en meget stor ballast fastgjort direkte til hovedarmen som energikilde. Da der ikke længere skal være plads til et mandskab til at trække i snorene under hovedarmen, bliver der plads til en sliske, i hvilket projektilet placeres inden affyringen. Slisken gjorde det muligt at gøre slyngen til projektilet længere, og derved forlænge kastelængden. En side gevinst ved slisken, er at under den første del af affyringen, hvor projektilet stadig er i slisken, er projektilet holdt fast i skudretningen, hvilket øger træfsikkerheden.



Figur 2: *Ballastens bevægelse under et skud. Illustration fra [Chevedden and Eigenbrod(1995)].*

Den næste store forbedring er at hængsle ballasten, dvs. hænge ballasten i en kurv under hovedarmen. Når ballasten er længst nede (og har frigivet mest muligt potentiel energi), er hovedarmen i bevægelse. Er ballasten fastspændt til hovedarmen, er noget af den frigivne potentielle energi bundet som bevægelsesenergi i ballasten. Ved at hængsle ballasten, giver man hovedarmen mulighed for at bevæge sig med en anden hastighed end ballasten, derved kan man undgå at ballasten binder al for meget bevægelsesenergi, der ellers

kunne være overført til projektilet. Se eksempel på bevægelsen af en hængslet ballast på figur 2. Virkningsgraden for en blide med en hængslet ballast er i computersimuleringer fundet til ca. 70%, når der tages højde for bl.a. hovedarmens masse[Chevedden and Eigenbrod(1995)].

I forbindelse med sine studier af bliden, opdagede Jordanus af Nemore (1225 – 1260) flere velkendte fysiske principper, bl.a.: (1) vægtstangsprincippet, (2) at arbejde er lig kraft gange vej og (3) potentiel energi i et tyngdefelt. Det menes, at penduluret er inspireret af bliden[Chevedden and Eigenbrod(1995)].

Bliden var kommet for at blive i mange år. Det var først 200 år efter kanonens fremkomst, at bliden blev udfaset, det skyldes, at de tidligere kanoner ikke var nær så pålidelige og effektive som en traditionel blide. Den sidste anvendelse af en blide i krigsmæssig sammenhæng er under den spanske belejring af Tenochtitlan i 1521. På grund af mangel på traditionel ammunition bygger General Cortes en blide. Bliden overlever dog ikke det første skud, da den skyder sig selv i sænk.

Bliden har sat mange spor i historien, og desuden kan enkelte ord spores direkte tilbage til bliden; det gælder f.eks. for fagbetegnelsen ingeniør:

> During their heyday, trebuchets received much attention from engineers – indeed, the very word "engineering" is intimately related to them. In Latin and the European vernaculars, a common term for trebuchet was "engine" (from ingenium,"an ingenious contrivance"), and those who designed, made and used them were called ingeniators. [Chevedden and Eigenbrod(1995)]

1.3 Blider i dag

I dag findes der flere rekonstruktioner af blider i stor størrelse. I Danmark findes der to blider på Middelaldercentret, der ligger på Lolland-Falster. Centrets største blide har en ballast på $2000kg$ og slynger projektiler, med en vægt på $15kg$, $169m$. Der skal ti – tolv mand til at betjene bliden, der kan affyre et projektil hvert tiende minut.

Rent videnskabeligt blev interessen for bliden øget kraftigt i 1995, da Scientific American gjorde bliden til en forsidehistorie [Chevedden and Eigenbrod(1995)]. Til baggrund for artiklen var flere computer–simuleringer, der viste, at bliden var overraskende effektiv, med mekaniske virkningsgrader på over 70%! Efterfølgende har flere andre f.eks. [Siano(2001)] arbejdet videre med blidens mekanik. Det er dette arbejde, der er udgangspunktet for dette projekt.

1.4 Mål med projektet

Hovedmålet med projektet er at undersøge blidens mekanik nærmere, herunder at finde en sammenhæng mellem blidens proportioner og den resulterende skudlængde samt det tidspunkt i hvilket projektilet skal slippes, for at komme så langt som muligt.

Undersøgelsen af blidens mekanik gribes an i nedenstående trin, og skal give en forståelse for, hvorfor konstruktionen er så effektiv til at omdanne potentiel energi bundet i ballasten, til kinetisk energi i projektilet.

- Modellering af systemet
- Opstil bevægelses ligninger for modellen
- Findes der en analytisk løsning
- Udvikling af en computermodel, der simulerer blidens bevægelser
- Test af computermodellen
- Undersøge den øvre grænse for blidens virkningsgrad.

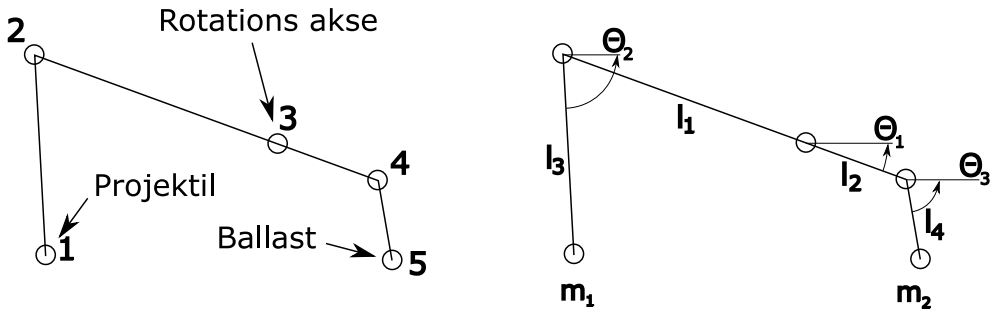
2 Modellering af systemet

Inden det er muligt at anvende metoder fra den klassiske mekanik, skal der opbygges en beregningsmodel af en blide. Udgangspunktet for en sådan model er bliden med hængslet ballast, se figur 1. Modellen er det system, som resten af opgaven handler om, så denne skal vælges med omhu, så der skal laves et kompromis mellem modellens detaljegrad og det beregningsarbejde, der er forbundet med at bestemme f.eks. bevægelsesligningerne for systemet.

Der er flere elementer, som er vitale for blidens funktion, de skal derfor medtages i modellen; det drejer sig om slyngposen med projektilet, hovedarmen og ballasten (inkl. ophængningen). Derudover er der elementer, som er af mindre betydning, det drejer sig om slisken, som styrer retningen af projektilet under den første del af en affyring og krogen, som afgør hvornår projektilet frigives fra slyngen, dette er ikke medtaget i modellen.

Slyngen er normalt udformet som to stykker reb, der går ned til en pose, som kan indeholde projektilet; under den første del af en affyring, bliver slyngen med projektilet trukket efter den ene ende af hovedarmen. Når der kun bliver trukket i længderetningen af et stykke reb, er det næsten identisk med en fast stang. I modellen bliver slyngen repræsenteret af en fast stang med en punktmasse (m_1) fastgjort i den ene ende.

Hovedarmen består af en del tømmer, hvis opgave er at agere vægtstang mellem ophængningspunktet for ballasten, og det sted hvor slyngen er fastgjort. I modellen repræsenteres hovedarmen som en fast og masseløs stang, der roterer omkring et fast punkt. Hovedarmen er langt fra masseløs, men pga. to ting giver det mening af se bort fra denne: (1) massen af armen er lille sammenlignet med ballasten og (2) hovedarmen bevæger sig meget langsommere end projektilet og indeholder derfor væsentligt mindre energi.



Figur 4: *Skitse af en blide med hængslet ballast, hvor de relevante størrelser er indtegnet. Det er denne model, der arbejdes videre med i resten af projektet. I tabel 1, er de enkelte størrelser nærmere forklaret.*

	Beskrivelse
m_1	Masse af projektilet
m_2	Masse af ballasten
l_1	Længde fra hovedaksel til projektilets ophængningspunkt.
l_2	Længde fra hovedaksen til ophængningspunkt for ballasten.
l_3	Længden af snoren ud til projektilet.
l_4	Længde fra ophængning af ballasten til ballastens massemidtpunkt.
θ_1	Hældning af hovedarm i forhold til vandret.
θ_2	Hældning af slyngen (projektil armen).
θ_3	Hældning af ballast armen.

Tabel 1: *Relevante størrelser for en blide. De tre nederste størrelser anvendes som generaliserede koordinater. På figur 4, er sammenhængen mellem de enkelte værdier vist grafisk.*

Ballasten er ophængt på hovedarmen. I modellen repræsenteres ballasten som en fast stang, der er fastgjort til hovedarmen i den ene ende og med en punktmasse (m_2) fastgjort i den anden ende.

En skitse af modellen er vist på figur 4, mens de tilhørende værdier er forklaret i tabel 1. I modellen ses der bort fra følgende:

- Massen i hovedarmen
- Elastiske egenskaber af hovedarm og slyngposen
- Interaktioner mellem projektilet og slisken
- Alle former for friktion (lejer og luftmodstand)

Ud fra modellen er vi interesserede i at bestemme den maksimale skudlængde, der kan opnås, givet at projektilet slippes på det rigtige tidspunkt. De parametre der er med til at påvirke modellens bevægelser er: blidens proportioner, massen af projektilet, ballasten og tyngde accelerationen, så det må forventes, at den maksimale kastelængde også afhænger af disse. Inden der anvendes klassisk mekanik på problemet, vil jeg undersøge, hvad man kan bestemme vha. dimensionsanalyse.

3 Dimensionsanalyse

Grundlaget for dimensionsanalysen (Buckinghams π–teorem) introduceres og det undersøges hvordan blidens kastelængde og skudtid skalerer.

3.1 Buckinghams π–teorem

π–teoremet tager udgangspunkt i at fysikken af et system ikke afhænger af hvilke enheder man benytter til at beskrive systemet [Price(2003)]. Følges denne konstatering til dørs ender man med Buckinghams π teorem [Buckingham(1914)]. Teoremet går i korte træk ud på følgende.

Givet en fysisk meningsfuld relation på formen:

$$0 = f(x_1, x_2, x_3, \ldots, x_n)$$

Så kan denne omskrives til en relation, der udelukkende benytter de enhedsløse størrelser (π_i), der kan dannes ud fra x_i, på denne måde:

$$\pi_i = \prod x_j^{a_j}$$

De valgte π_i skal være uafhængige af hinanden, det kan f.eks. sikres ved, at der i alle de dimensionsløse størrelser π_i, indgår en af de oprindelige størrelser x_j, som ikke indgår i de andre dimensionsløse størrelser. Den fysiske relation kan nu skrives som:

$$0 = F(\pi_1, \pi_2, \ldots, \pi_k)$$

Hvor antallet af parametre i den enhedsløse relation (k) opfylder:

$$k = n - r$$

Hvor r er antallet af grundlæggende enheder i de oprindelige værdier (f.eks. meter, kilogram og sekunder).

3.2 π teoremet anvendt på bliden

Vi tager udgangspunkt i en blide, der fastholdes i en stilling. Ved start af bliden fjernes de bånd, der fastholder bliden i denne stilling. Det betyder, at alle begyndelseshastigheder er nul, blidens start–tilstand kan så entydigt beskrives ved følgende værdier:

- Ballast og projektil masse (enhed masse): m_1 og m_2
- Længder af de forskellige dele af bliden (enhed længde): l_1, l_2, l_3 og l_4.
- Vinkler af de forskellige led (dimensionsløs): θ_1, θ_2, θ_3

Den fysiske mekanisme der driver bliden er tyngdekraften, så tyngdeaccelerationen g er nødvendig for at beskrive bevægelsen.

3.2.1 Hvor langt kan man kaste?

Givet dimensionerne og udgangsstillingen for en blide, ønsker jeg, at bestemme hvor langt bliden kan kaste, hvis projektilet frigives på tidspunktet for maksimal kastelængde. Kastelængden kaldes for R og den må være givet ved en relation på følgende form:

$$0 = f(R, m_1, m_2, l_1, l_2, l_3, l_4, \theta_1, \theta_2, \theta_3, g)$$

Vha. π–teoremet kan antallet af parametre i relationen reduceres fra 11 til 8, da der i de oprindelige parametre indgår tre basis enheder (masse, længde og tid). Jeg har valgt at benytte størrelsen af ballasten m_2 som vægt–reference og længden fra hovedakslen til ballastens ophængningspunkt l_2 som længde reference[1]. Den nye relation, med de valgte referencer, kan f.eks. se ud som følger:

$$0 = F\left(\frac{R}{l_2}, \frac{m_1}{m_2}, \frac{l_1}{l_2}, \frac{l_3}{l_2}, \frac{l_4}{l_2}, \theta_1, \theta_2, \theta_3\right) \tag{3.1}$$

Bemærk at tyngdeaccelerationen *ikke* indgår i udtrykket, hvilket betyder, at kastelængden er uafhængig af denne. Desuden ses, at skaleres alle længderne af bliden op, med en given faktor, vil bliden skyde en tilsvarende faktor længere.

3.2.2 Hvornår skal der gives slip på projektilet?

Givet en udgangsstilling, hvornår skal bliden så slippe projektilet, for at det kommer så langt som muligt? Det rigtige tidspunkt kan kaldes t og indgår i relationen:

$$0 = g(t, m_1, m_2, l_1, l_2, l_3, l_4, \theta_1, \theta_2, \theta_3, g)$$

Igen kan antallet af parametre reduceres med 3; en ny relation kan se ud som denne:

$$0 = G\left(\frac{t^2 g}{l_2}, \frac{m_1}{m_2}, \frac{l_1}{l_2}, \frac{l_3}{l_2}, \frac{l_4}{l_2}, \theta_1, \theta_2, \theta_3\right) \tag{3.2}$$

Den første af funktionens variable fortjener lidt ekstra opmærksomhed, $\frac{t^2 g}{l_2}$. Ud fra denne giver det god mening at indføre en karakteristisk tid (τ) for systemet, givet ved:

$$\tau = \sqrt{l_2/g}$$

Vi er interesserede i et sving med hovedarmen. Tidsskalaen defineres derfor ud fra længden mellem omdrejningspunktet og ophængningspunktet for ballasten, denne længde er l_2.

[1] Et sådant valg kan gøres for at bringe noget mere håndgribelig fysik ind i systemet. Valget af l_2 som længde reference kan begrundes med at $\sqrt{l_2/g}$ er en karakteristisk tid for hovedarmens svingninger.

4 Udledning af bevægelsesligningerne

For at undersøge bevægelsen af bliden skal der opstille nogle ligninger, der beskriver sammenhængen mellem blidens nuværende tilstand (beskrevet ved et antal positioner og hastigheder) og hvordan denne tilstand udvikles over tid. Disse ligninger kaldes for bevægelsesligninger og kan bestemmes på flere måder, jeg vil benytte en metode baseret på Lagrange funktionen. Der findes en metode, der tager udgangspunkt i Hamilton funktionen, men beregningerne til Hamilton metoden bliver ofte længere end de tilsvarende for Lagrange metoden. Det giver ikke problemer at fravælge Hamilton metoden, da denne giver et resultat, der svarer til resultatet fra Lagrange metoden (dynamikken af det fysiske system er ens). Hamilton metoden vil for den aktuelle model give seks koblede første ordens differentialligninger, mens Lagrange metoden giver tre anden ordens differentialligninger (der kan omskrives til seks førsteordens ...).

[Wraa(1997)] Inden vi kan finde Lagrange funktionen, er det nødvendig at finde et sæt generaliserede koordinater, der er et sæt af størrelser, der entydigt kan beskrive systemets tilstand fuldstændigt mht. stilling. På figur 4 er vist den model af en blide, jeg vil arbejde ud fra. For en given model er m_1, m_2, l_1, l_2, l_3 og l_4 parametre, der bestemmes, idet modellen konstrueres, mens de værdier, der *ændres* under en bevægelse, θ_1, θ_2 og θ_3, kan benyttes som generaliserede koordinater. Til de generaliserede koordinater er knyttet de generaliserede hastigheder: $\dot{\theta}_1$, $\dot{\theta}_2$ og $\dot{\theta}_3$.

4.1 Lagrange funktionen

Lagrange funktionen er defineret ved systemets kinetiske energi fratrukket systemets potentielle energi:

$$
\begin{aligned}
L(\vec{q},\dot{\vec{q}}) &= E_{kin}(\dot{\vec{q}},\vec{q}) - E_{pot}(\vec{q}) \\
E_{kin} &= \frac{1}{2}m_1(\dot{x}_1^2+\dot{y}_1^2) + \frac{1}{2}m_2(\dot{x}_5^2+\dot{y}_5^2) \\
E_{pot} &= m_1 g y_1 + m_2 g y_5
\end{aligned}
$$

For at opstille udtrykket for Lagrange funktionen skal de kartesiske koordinater udtrykkes ved de generaliserede koordinater, disse udtryk kan så direkte indsættes i de kendte udtryk for kinetisk og potentiel energi:

$$
\begin{aligned}
x_1 &= -l_1\cos\theta_1 + l_3\cos\theta_2 & \qquad y_1 &= -l_1\sin\theta_1 + l_3\sin\theta_2 \\
x_5 &= l_2\cos\theta_1 + l_4\cos\theta_3 & \qquad y_5 &= l_2\sin\theta_1 + l_4\sin\theta_3
\end{aligned}
$$

I udtrykket for den kinetiske energi anvendes summen af kvadratet af de kartesiske hastigheder $\dot{x}_i^2+\dot{y}_i^2$, der for de to punktmasser er givet ved[2]:

$$
\begin{aligned}
\dot{x}_1^2+\dot{y}_1^2 &= l_1^2\dot{\theta}_1^2 + l_3^2\dot{\theta}_2^2 - 2l_1l_3\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2) \\
\dot{x}_5^2+\dot{y}_5^2 &= l_2^2\dot{\theta}_1^2 + l_4^2\dot{\theta}_3^2 + 2l_2l_4\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)
\end{aligned}
$$

[2]Til en generaliseret koordinat q_i hører der en generaliseret hastighed $\dot{q}_i$, der er givet ved den tidsafledte af den generaliserede koordinat: $\dot{q}_i = \frac{dq_i}{dt}$

Lagrange funktionen er så givet ved følgende udtryk:

$$\begin{aligned} L = &\frac{1}{2}m_1\left(l_1^2\dot{\theta}_1^2 + l_3^2\dot{\theta}_2^2 - 2l_1l_3\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\right) \\ &+ \frac{1}{2}m_2\left(l_2^2\dot{\theta}_1^2 + l_4^2\dot{\theta}_3^2 + 2l_2l_4\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\right) \\ &- (m_2l_2 - m_1l_1)g\sin\theta_1 - m_1gl_3\sin\theta_2 - m_2gl_4\sin\theta_3 \end{aligned} \tag{4.1}$$

4.2 Opstil bevægelsesligningerne

Ud fra Lagrange funktionen, kan der direkte opskrives [Wraa(1997), side 5] et udtryk for hver af de generaliserede koordinater, der relaterer de generaliserede accelerationer til de generaliserede koordinater og hastigheder:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial\dot{\theta}_i}\right) = \frac{\partial L}{\partial\theta_i} \tag{4.2}$$

De udregninger der er involveret fylder en del, og jeg vil derfor kun udlede udtrykket for θ_1, de to andre udtryk vil blive gengivet efter udledningen.

Jeg tager udgangspunkt i at evaluere de enkelte led i (4.2):

$$\begin{aligned} \frac{\partial L}{\partial\theta_1} &= m_1l_1l_3\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2) - m_2l_2l_4\dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3) \\ &\quad - (m_2l_2 - m_1l_1)g\cos\theta_1 \\ \frac{\partial L}{\partial\dot{\theta}_1} &= m_1l_1^2\dot{\theta}_1 - m_1l_1l_3\dot{\theta}_2\cos(\theta_1-\theta_2) \\ &\quad + m_2l_2^2\dot{\theta}_1 + m_2l_2l_4\dot{\theta}_3\cos(\theta_1-\theta_3) \end{aligned} \tag{4.3}$$

Når den absolutte tidsafledte af det ovenstående udtryk skal bestemmes, skal man huske, at både θ_i og $\dot{\theta}_i$ er tidsafhængige.

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial\dot{\theta}_1}\right) = &(m_1l_1^2 + m_2l_2^2)\ddot{\theta}_1 - m_1l_1l_3\ddot{\theta}_2\cos(\theta_1-\theta_2) \\ &+ m_1l_1l_3\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2) - m_1l_1l_3\dot{\theta}_2^2\sin(\theta_1-\theta_2) \\ &+ m_2l_2l_4\ddot{\theta}_3\cos(\theta_1-\theta_3) - m_2l_2l_4\dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3) \\ &+ m_2l_2l_4\dot{\theta}_3^2\sin(\theta_1-\theta_3) \end{aligned} \tag{4.4}$$

Til sidst skal højresiden af (4.3) og (4.4), sættes lig hinanden hvorefter alle led der indeholder noget med acceleration samles på venstre side og resten på højre side:

$$\begin{aligned} &(m_1l_1^2 + m_2l_2^2)\ddot{\theta}_1 - m_1l_1l_3\ddot{\theta}_2\cos(\theta_1-\theta_2) \\ &+ m_2l_2l_4\ddot{\theta}_3\cos(\theta_1-\theta_3) = -(m_2l_2 - m_1l_1)g\cos\theta_1 \\ &\qquad + m_1l_1l_3\dot{\theta}_2^2\sin(\theta_1-\theta_2) - m_2l_2l_4\dot{\theta}_3^2\sin(\theta_1-\theta_3) \end{aligned} \tag{4.5}$$

For θ_2 og θ_3 bliver de tilsvarende udtryk:

$$l_3\ddot{\theta}_2 - l_1\ddot{\theta}_1\cos(\theta_1-\theta_2) = -g\cos\theta_2 - l_1\dot{\theta}_1^2\sin(\theta_1-\theta_2) \tag{4.6}$$

$$l_4\ddot{\theta}_3 + l_2\ddot{\theta}_1\cos(\theta_1-\theta_3) = -g\cos\theta_3 + l_2\dot{\theta}_1^2\sin(\theta_1-\theta_3) \tag{4.7}$$

For at bestemme vinkelaccelerationerne, kan (4.5), (4.6) og (4.7) skrives op som et linært ligningssystem:

$$a\ddot{\theta}_1 + b\ddot{\theta}_2 + c\ddot{\theta}_3 = d \qquad e\ddot{\theta}_1 + f\ddot{\theta}_2 = g \qquad h\ddot{\theta}_1 + i\ddot{\theta}_3 = j$$

der kan løses vha. kendte metoder fra linær algebra. Koefficienterne til ligningssystemet $(a, b, \ldots, j)$ er bestemt ud fra θ_i og $\dot{\theta}_i$.

5 Forsøg på analytisk løsning

Når man gerne vil undersøge et fysisk system, er det optimalt at finde en analytisk løsning, sådan at man ud fra et sæt startbetingelser (udgangsstilling), kan bestemme systemets position til en given efterfølgende tid.

For at prøve at finde en sådan løsning vil jeg benytte Hamilton–Jacobi metoden.

5.1 Hamilton–Jacobi metoden

Når man skal anvende Hamilton–Jacobi metoden, består den af to dele: først skal man ud fra Hamilton funktionen opstille en differential ligning, for en virkningsfunktion S (dette er det fysiske skridt). Dette er Hamilton–Jacobi ligningen, der er vist herunder:

$$0 = \frac{\partial S}{\partial t} + H\left(q_1, \ldots, q_s, \frac{\partial S}{\partial q_1}, \ldots, \frac{\partial S}{\partial q_s}, t\right)$$

Det næste skridt er at finde en løsning, der tilfredsstiller Hamilton–Jacobi ligningen. Til det anvendes en teknik baseret på separation af variable, der kaldes for Hamilton–Jacobi metoden. På bilag D, er Hamilton–Jacobi ligningen og Hamilton–Jacobi metoden introduceret.

5.2 Hamilton funktionen

Hamilton funktionen er en Legendre transformation af Lagrange funktionen. For at kunne opstille Hamilton funktionen, skal der findes et udtryk for de generaliserede momenta (p_i), der er givet ved Lagrange funktionen. De generaliserede momenta er så givet ved:

$$\begin{aligned} p_1 = \frac{\partial L}{\partial \dot{\theta}_1} &= m_1 l_1^2 \dot{\theta}_1 - m_1 l_1 l_3 \dot{\theta}_2 \cos(\theta_1 - \theta_2) \\ &\quad + m_2 l_2^2 \dot{\theta}_1 + m_2 l_2 l_4 \dot{\theta}_3 \cos(\theta_1 - \theta_3) \end{aligned} \tag{5.1}$$

$$p_2 = \frac{\partial L}{\partial \dot{\theta}_2} = m_1 l_3^2 \dot{\theta}_2 - m_1 l_1 l_3 \dot{\theta}_1 \cos(\theta_1 - \theta_2) \tag{5.2}$$

$$p_3 = \frac{\partial L}{\partial \dot{\theta}_3} = m_2 l_4^2 \dot{\theta}_3 + m_2 l_2 l_4 \dot{\theta}_1 \cos(\theta_1 - \theta_3) \tag{5.3}$$

Ligningssystemet for θ_1, θ_2 og θ_3 løses:

$$\begin{pmatrix} m_1 l_1^2 + m_2 l_2^2 & -m_1 l_1 l_3 \cos(\theta_1 - \theta_2) & m_2 l_2 l_4 \cos(\theta_1 - \theta_3) \\ -m_1 l_1 l_3 \cos(\theta_1 - \theta_2) & m_1 l_3^2 & 0 \\ -m_2 l_2 l_4 \cos(\theta_1 - \theta_3) & 0 & m_2 l_4^2 \end{pmatrix} \cdot \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

Løses ligningssystemet mht. $\dot{\theta}_i$, fås følgende[3]:

[3] Løsningen er foretaget på computer i programmet maple.

$$\dot{\theta}_1 = -\frac{l_3 l_4 p_1 + l_4 l_1 \cos(\theta_1 - \theta_2) p_2 - l_3 l_2 \cos(\theta_1 - \theta_3) p_3}{l_3 l_4 \left(-m_1 l_1^2 - m_2 l_2^2 + l_2^2 (\cos(\theta_1 - \theta_3))^2 m_2 + l_1^2 (\cos(\theta_1 - \theta_2))^2 m_1\right)}$$

$$\dot{\theta}_2 = -\frac{p_2 l_4 m_1 {l_1}^2 + p_2 l_4 m_2 {l_2}^2 - p_2 l_4 {l_2}^2 (\cos(\theta_1 - \theta_3))^2 m_2 + m_1 l_1 \cos(\theta_1 - \theta_2) l_3 l_4 p_1 - m_1 l_1 \cos(\theta_1 - \theta_2) l_3 l_2 \cos(\theta_1 - \theta_3) p_3}{m_1 {l_3}^2 l_4 \left(-m_1 {l_1}^2 - m_2 {l_2}^2 + {l_2}^2 (\cos(\theta_1 - \theta_3))^2 m_2 + {l_1}^2 (\cos(\theta_1 - \theta_2))^2 m_1\right)}$$

$$\dot{\theta}_3 = \frac{-p_3 l_3 m_1 {l_1}^2 - p_3 l_3 m_2 {l_2}^2 + p_3 l_3 {l_1}^2 (\cos(\theta_1 - \theta_2))^2 m_1 + m_2 l_2 \cos(\theta_1 - \theta_3) l_3 l_4 p_1 + m_2 l_2 \cos(\theta_1 - \theta_3) l_4 l_1 \cos(\theta_1 - \theta_2) p_2}{m_2 {l_4}^2 l_3 \left(-m_1 {l_1}^2 - m_2 {l_2}^2 + {l_2}^2 (\cos(\theta_1 - \theta_3))^2 m_2 + {l_1}^2 (\cos(\theta_1 - \theta_2))^2 m_1\right)}$$

Det er ikke muligt at separere (p_i, θ_i) som en funktion, der ikke indeholder andre værdier af p_i eller θ_i, det skyldes at der er led på formen $a_1 p_2 + a_2 \cos(\theta_1 - \theta_2)$. Det er derfor heller ikke muligt at lave separation af variable, og det virker ikke sandsynligt at komme videre med Hamilton–Jacobi metoden. At det ikke er muligt at separere de variable størrelser udelukker hverken, at der findes en analytisk løsning eller at der er andre bevarede størrelser for systemet end energien.

5.3 Argument fra beregnelighedsteori

Inden for den teoretiske datalogi, har man en disciplin, der arbejder med hvad der er muligt at beregne og hvad der ikke giver mening af regne på [Lewis and Papadimitriou(1998)]. Først er der et eksempel på en metode, der anvendes, til at vise at noget ikke kan beregnes.

Et af de klassiske resultater fra faget "Beregnelighed"[4] er, at man ikke kan afgøre om en Turing maskine stopper på et givet input, uden at køre Turing maskinen på inputtet [Lewis and Papadimitriou(1998)]. En Turing maskine er en teoretisk konstruktion, der kan beregne det samme som en moderne computer. Dette resultat kaldes Halting problemet og er generelt uberegneligt. At noget er generelt uberegneligt forhindrer ikke, at det kan afgøres i specifikke tilfælde.

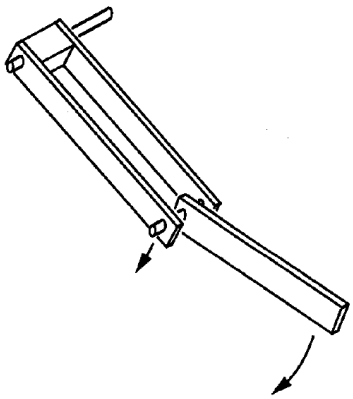
> Kan det afgøres om to vilkårlige Turing maskiner begge stopper på et givent input?

Metode: Først antages det, at der eksisterer en metode til at afgøre problemet. Dernæst reduceres problemet til et problem, vi ved ikke kan løses. Det gøres ved at sætte en af de vilkårlige Turing maskiner til altid at stoppe uanset input. Om begge Turing maskiner stopper på et givent input, afhænger nu af om *en* Turing maskine stopper på et givent input; dette problem er Halting problemet, som er uløseligt. Hvis der findes en metode til at løse det oprindelige spørgsmål, så kan vi løse Halting problemet. Dette er i modstrid med Halting problemet og antagelsen må være forkert.

> Der findes *ikke* en metode til at afgøre om to vilkårlige Turing maskiner stopper på et givent input.

Et tilsvarende argument kan man lave med fysiske systemer. Her kan man sige noget om, hvorvidt der findes en analytisk løsning til et givet fysisk system.

[4] Automatteori og beregnelighed, DM17 på SDU Odense.



Figur 5: *Skitse af et dobbelt pendul / kaotisk pendul, som er undersøgt af [Shinbrot et al.(1992)Shinbrot, Grebogi, Wisdom, and Yorke]. Illustrationen er fra samme sted.*

> Kan et system reduceres (simplificeres) til et system, der *ikke* kan beregnes / løses analytisk, kan det oprindelige system heller ikke beregnes / løses analytisk.

Antag at der findes en analytisk løsning til blidens bevægelse, så må denne beskrivelse også gælde, når den ene masse (f.eks. projektilet) går mod nul. Nu er beskrivelsen af bliden identisk med et kaotisk pendul, for hvilket vi ved, at der ikke findes en analytisk løsning[5]. Dette er en modstrid, så antagelsen er forkert, og der findes derfor ikke en analytisk løsning til blidens bevægelse.

Der er forsøgt at finde en analytisk løsning til bevægelsesligningerne vha. Hamilton–Jacobi metoden, det lykkes ikke, da det ikke er muligt at lave separation af variable på et bestemt udtryk. At Hamilton–Jacobi metoden fejler, udelukker ikke, at der findes en analytisk løsning til bevægelsesligningerne.

Til at sandsynliggøre at der ikke findes en analytisk løsning, benyttes et argument fra beregnelighedsteori. Resultatet er, at hvis der findes en analytisk løsning til blidens bevægelse, kan denne løsning reduceres til et kaotisk pendul, som sandsynligvis ikke har en analytisk løsning.

[5] [Shinbrot et al.(1992)Shinbrot, Grebogi, Wisdom, and Yorke] viser eksperimentelt og numerisk at dobbelt pendulet er følsomt over for start betingelserne, dvs. at det udviser kaotisk opførsel. Der har ikke været muligt at finde referencer der siger at et kaotisk system ikke kan have en analytisk løsning; jeg har så snakket med Dr. Scient Paolo Sibani og Dr. Scient Sven Tougaard, begge fra Institut for Fysik og Kemi SDU, der begge mener at der ikke findes analytiske løsninger til kaotiske systemer.

6 Udarbejdelse af computersimulering

I kapitlet beskrives valg af værktøjer til udarbejdelsen af implementationen; python som programmerings sprog og en fjerde ordens Runge–Kutta metode som diskretiserings metode. Så beskrives det hvordan programmet er blevet testet, bl.a. vha. bestemmelse af de anvendte metoders orden ud fra simulerings data.

6.1 Valg af værktøjer

Da det ikke er muligt at finde et analytisk udtryk for blidens bevægelser, må vi lede efter andre metoder til at undersøge blidens bevægelse. Efter en analytisk løsning må det næstbedste være at approksimere bevægelsen vha. en numerisk løsning af bevægelsesligningerne. Det kan gøres på flere forskellige måder, jeg har valgt at benytte en fjerde ordens Runge-Kutta metode, med den begrundelse at metoden er absolut stabil[6] og at den giver et godt forhold mellem den opnåede præcision og antallet af beregninger.

Jeg har valgt at skrive programmet i sproget Python primært af to grunde: (1) det er let at skrive kode i python og (2) Python kode er let at læse også for ikke programmører. Det sker dog på bekostning af hastigheden af det endelige program, da python er væsentligt langsommere end sprog som C++ eller Fortran[7].

Programmet virker ved, i hvert skridt, at evaluere vinkelaccelerationerne $\ddot{\theta}_i$ ud fra bevægelsesligningerne og derefter fremskrive værdierne af $\dot{\theta}_i$ og θ_i, til en ny tilstand, dette gentages indtil et givet tidskriterie er opfyldt. På denne måde bestemmes det, hvordan de tre vinkler udvikler sig over tid, se et eksempel på en sådan udvikling på figur 9.

For at få nogle flere informationer ud af simulationen, beregner programmet nogle flere størrelser efter hvert skridt. Det drejer sig om den kinetiske energi og den potentielle energi af projektilet og ballasten, desuden beregner programmet kastelængden for projektilet (under de antagelser at projektilet bliver frigivet i dette øjeblik, og at der ses bort fra luftmodstand.).

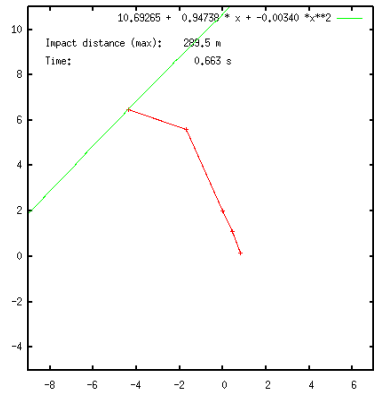
For at gøre det lettere at undersøge virkemåden af programmet, er der implementeret mulighed for at vise øjebliks billeder fra simuleringen. Reelt virker dette som en slowmotion–optagelse af den skydende blide. For at gøre det let at se hvordan programmet virker, er der lagt videoer af programmet i funktion op på adressen: `http://www.fys.sdu.dk/fysikshow/blide/`. Koden til programmet ligger også på den samme side, inkl. en beskrivelse af hvordan man får programmet til at virke på sin egen computer. Se desuden appendix A, hvor der er givet eksempler på brugen af programmet.

[6] I følge [Christiansen(1999), side 142]: At metoden formindsker den udbredte fejl i hvert skridt.

[7] Denne forskel skyldes at Python er et fortolket sprog (dvs. at når programmet køres, er der et andet program der læser det linie for linie og gør hvad der står i koden), mens Fortran og C++ er oversatte sprog, hvor koden bliver oversat til et program der direkte kan køres af processoren.

6.2 Test af programmet

Inden vi begynder at undersøge blidens bevægelse med udgangspunkt i programmet, skal det tjekkes om programmet opfører sig på en fornuftig måde. Der er tjekket for tre forskellige ting: (1) den animerede bevægelse virker fysisk fornuftig (den store masse får hovedarmen til at svinge rundt, der igen får slyngposen til at accelerere, osv.), (2) at skudlængden konvergerer mod en fast værdi når skridtlængden mindskes og (3) at systemets samlede mekaniske energi er bevaret.



Figur 6: *Øjebliks billede fra en simulering, der ved dette tidspunkt viser en fysisk forkert opførsel, da projektilet via slyngposen skubber på hovedarmen.*

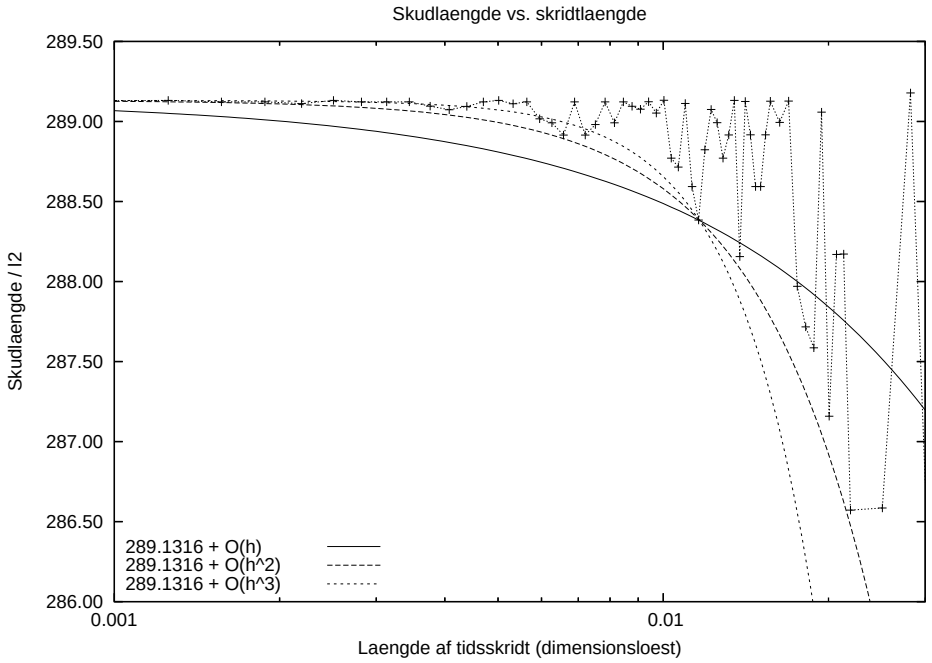
Det er svært at beskrive, hvornår en animation ser fysisk rigtigt ud, så det vil jeg undlade. Umiddelbart virker simuleringerne meget fornuftige[8], men omkring 0.7 sekund inde i simuleringen, sker der noget, der kan virke forkert, se billede fra simuleringen på figur 6. På dette tidspunkt skal ballasten til at starte en sidelæns bevægelse, dette kræver energi, som hentes ved at bremse hovedarmens rotation, samtidig ser det ud som om, at slyngen skubber på hovedarmen. I virkeligheden vil man forvente at slyngen krølles sammen, da den primært består af tovværk. Fejlen skyldes, at det under udledningen af bevægelsesligningerne antages at projektilet er forbundet til hovedarmen med en fast stang. Denne antagelse er kun god så længe der hives i rebet, og simuleringen fejler derfor på det beskrevne sted i simulationen.

På figur 7 er afbilledet sammenhængen mellem den fundne skudlængde og den anvendte skridtlængde. Det ses, at når skridtlængden mindskes konvergerer skudlængden mod en fast værdi (~ 289.13). I resten af afsnittet arbejdes der udelukkende med dimensionsløse størrelser, de anvendte skalaer bliver dog først introduceret i kapitel 7. Ved at indtegne forskellige fejlfunktioner på figuren, kan fejlen på metoden findes til at være begrænset af $O(h^2)$, Se en mere detaljeret forklaring i appendix B.

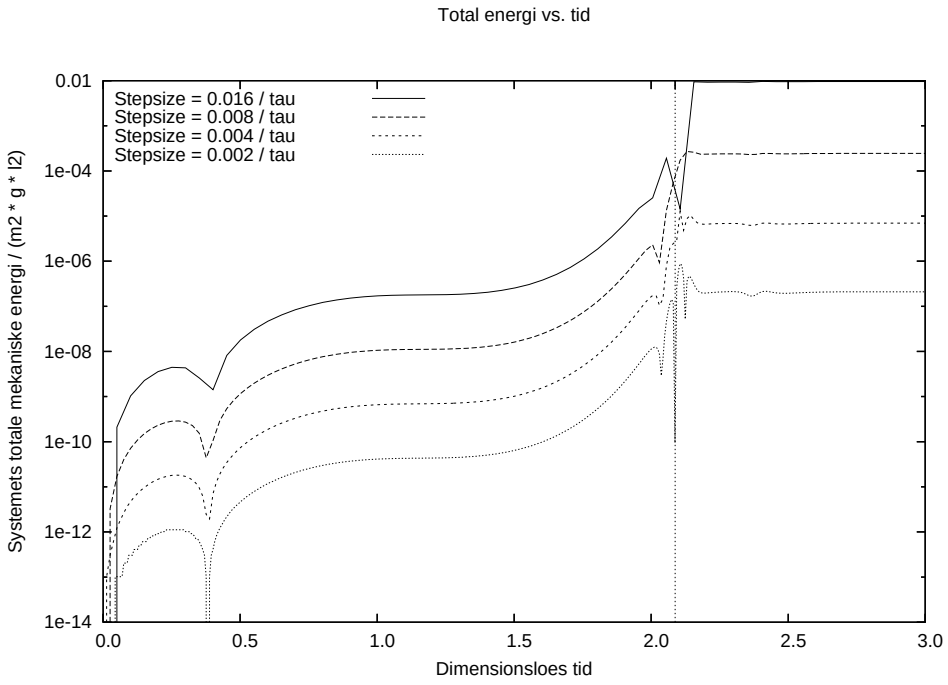
Ud fra de fundne resultater giver det ikke mening at benytte Richardson ekstrapolation[Christiansen(1999)], det skyldes at den dominerende fejl *ikke* er en systematisk metode fejl, som en ekstrapolation kan minimere. Det kan ses ved at resultaterne er meget svingende for de store skridtlængder. Richardson ekstrapolationen er kort beskrevet på appendix C.

Det sidste, der undersøges, er, hvordan systemets totale mekaniske energi ændres over tid. Fysikken fortæller os, at denne størrelse er bevaret, da der kun tages højde for konservative kræfter, og at Hamilton funktionen er tidsuafhængig. Da det er en numerisk løsning, kan det ikke forventes, at systemets totale energi er helt konstant, da afrundings og metode fejl ikke kan undgås. Ændringen af systemets totale mekaniske energi kan benyttes som en indikator for hvor stor en fejl, der er i simuleringen. På figur 8 er systemets totale mekaniske energi (fremover kaldet fejlen) afbilledet som funktion af tiden, for forskellige skridtlængder. På figuren skal følgende bemærkes: (1) at fejlen er proportional med den anvendte skridtlængde i fjerde potens og (2) at den største del af fejlen sker

[8] Video 1-5 viser hvad en ændring af slyngposens længde betyder for blidens dynamik og den resulterende skudlængde.



Figur 7: *Illustration af sammenhængen mellem den benyttede skridtlængde og den skudlængde der bestemmes. Det ses at fejlen er begrænset af $O(h^2)$, og at den benyttede metode derfor er en anden ordens metode.*



Figur 8: *Afbildning af den samlede energi i systemet gennem en simulering. De separate kurver svarer til forskellige skridtlængder. To ting skal bemærkes, kurverne for de forskellige skridtlængder minder rigtig meget om hinanden, og at ændringen i den samlede energi mindskes med ca. en faktor 16, når skridtlængden halveres.*

på et meget lille tidsrum. (1) er svært at se direkte ud fra grafen, dette er fundet ved at sammenligne fejlen til tiden 1.25 (dimensionsløs tid, se værdier i tabel 2), det ses at forholdet mellem fejlen er proportional med h^4.

h	ΔE	Forhold
0.0501	$1.2 \cdot 10^{-8}$	
0.0251	$7.4 \cdot 10^{-10}$	16.1
0.0125	$4.6 \cdot 10^{-11}$	16.0
0.0063	$2.9 \cdot 10^{-12}$	16.2

Tabel 2: *Fejl og længde af tidsskridt, til tiden* 1.25*. Forholdet mellem fejlen for to forskellige skridtlængder er også bergenet.*

Ud fra figur 7 og 8 kan det ses, at det ikke vil give problemer med de opnåede resultater, hvis der benyttes en skridtlængde, der er mindre end 0.01. Ved en skridtlængde på 0.01 er ændringen i den totale energi af størrelsesordnen 10^{-5}.

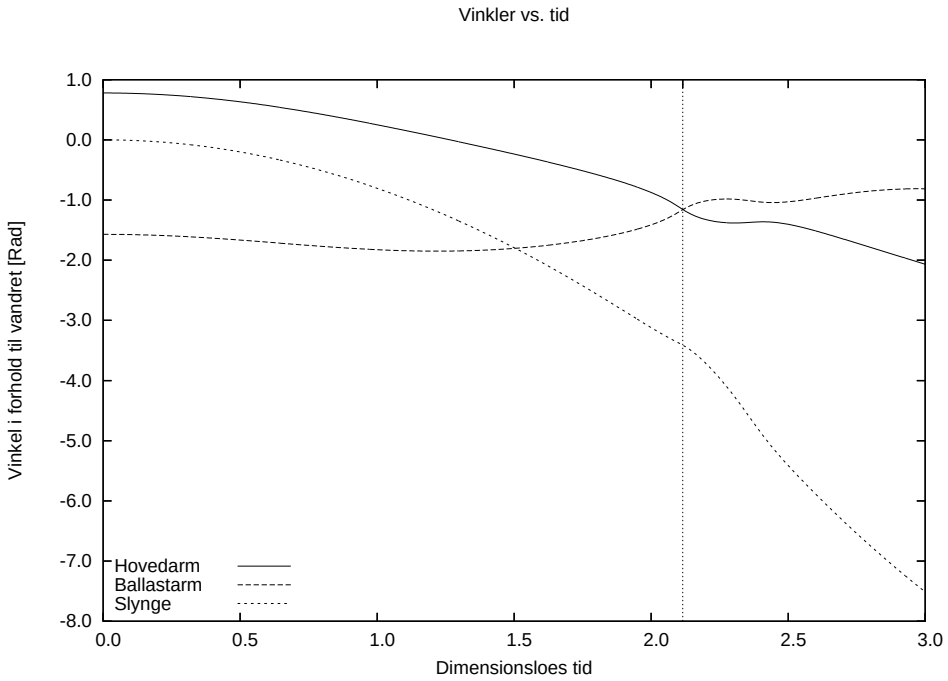
Jeg har fundet at: simuleringen virker fysisk rigtig, at kastelængden konvergerer mod en fast værdi, når skridtlængden mindskes og at systemets totale mekaniske energi er så godt som konstant for tilstrækkeligt små skridtlængder. De tre ting gør, at jeg roligt kan stole på resultatet af simuleringen.

6.3 Mulige forbedringer

Når man skal simulere noget, er man interesseret i to ting: at fejlen er så lille som muligt og at simulationen bliver færdig inden for en overskuelig tid.

Hvis man ønsker, at fejlen skal være så lille som muligt, kan man bare vælge en tilpas lille skridtlængde, problemet med denne metode er, at når skridtlængden halveres, skal der tages dobbelt så mange skridt og simulationen tager derfor dobbelt så lang tid. Figur 8 viser at fejlen vokser med forskellig hastighed afhængig af hvor i simuleringen man er. En god afvejning mellem fejl og tid vil så være at benytte store skridtlængder, når fejlen vokser langsomt, og små skidtlængder når fejlen vokser hurtigt.

Ved at skifte Python ud med et oversat sprog som C/C++ eller Fortran, kan køretiden reduceres med ca. en faktor 20. Denne værdi er fundet ved at oversætte et python program (en tidlig udgave af den implementerede simulering) til C++, og derefter sammenligne ydelsen af de to programmer.



Figur 9: *Afbildning af hvordan vinklerne ændres over tid for den optimerede blide. Tidspunktet for hvornår ballasten begynder at bremse hovedarmens bevægelse, svarer til skæringen mellem grafen for hovedarmen og ballasten. Til dette tidspunkt ligger ballasten som en direkte forlængelse af hovedarmen, og hovedarmen skal nu til at trække ballasten sidelæns, hvilket koster energi, for at kunne fortsætte sin bevægelse. Dette tidspunkt svarer præcist til tidspunktet for den maksimale skudlængde.*

7 Eksperimenter på simuleringen

Der er nu implementeret en computersimulering af vores model af en blide, og opgaven er nu at prøve at forstå / fortolke nogle af de fænomener som simuleringen giver os mulighed for at undersøge.

For at gøre figurerne mere generelle er alle de afbillede størrelser dimensionsløse. Den naturlige længde er l_2, da denne er med til at bestemme tiden for en svingning af hovedarmen og den i systemet tilgængelige energi. De anvendte skalaer er vist i tabel 3. I det følgende tager jeg udgangspunkt i en "standard" blide, med dimensioner som angivet i tabel 4.

Længde	$l_2 = 1m$
Tid:	$\tau = \sqrt{\frac{l_2}{g}} = 0.3191s$
Energi:	$\varepsilon = m_2 \cdot g \cdot l_2 = 982J$

Tabel 3: *Naturlige længde, tid og energiskalaer for systemet.*

I bilag B er der medtaget ekstra forklaringer til de enkelte figurer.

Det direkte resultat af simuleringen, er hvordan de tre generaliserede koordinater udvikler sig over tid; dette er vist på figur 9. Ud fra figuren er det svært at sige noget om hvor effektiv bliden er til at omdanne beliggenhedsenergi til bevægelsesenergi.

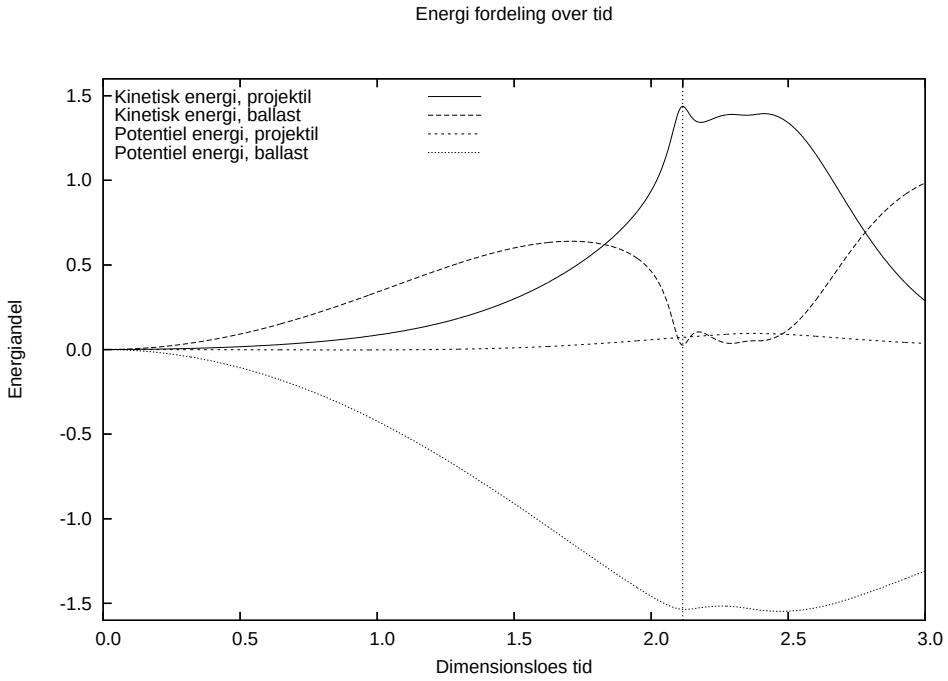
Størrelse	Værdi
m_1	$1kg$
m_2	$100kg$
l_1	$4m$
l_2	$1m$
l_3	$2.8m$
l_4	$1m$
θ_1	0.78
θ_2	0.0
θ_3	-1.57

Tabel 4: *Dimensioner for en standard blide.*

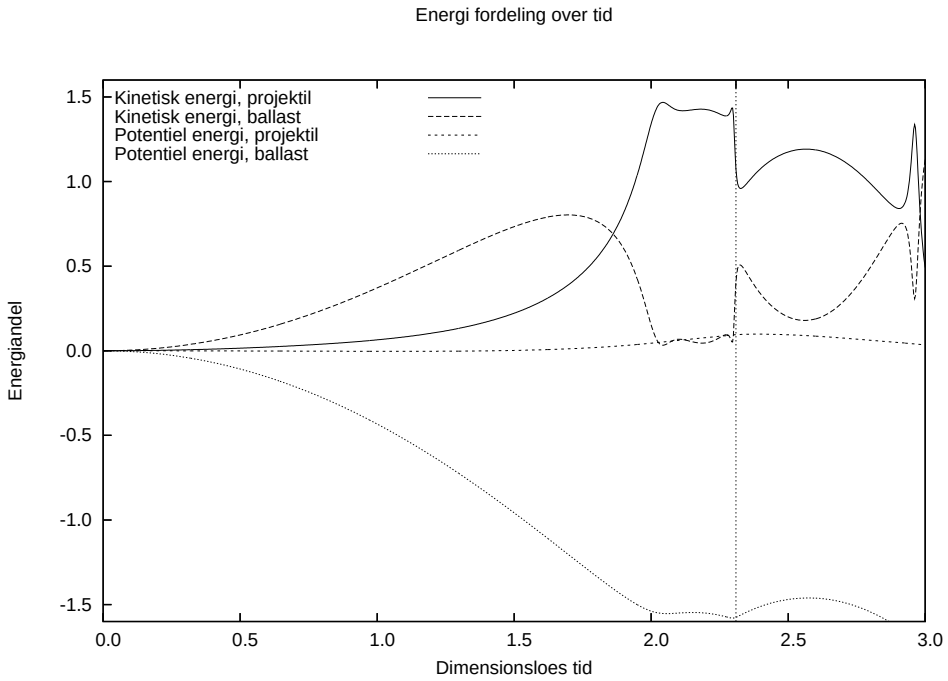
Ud fra simuleringen er det let at hente information om hvor meget energi der er bundet forskellige steder i systemet. Jeg ønsker at undersøge forskellen mellem en godt og en dårligt proportioneret blide, hhv. en standard blide og en standard blide med slyngposen forlænget en meter. For at undersøge hvordan energien flyttes rundt i de to systemer, er det på figur 10 og 11 vist hvor i systemerne energien er til et givet tidspunkt. Inden affyringen (de lodrette linier), ser man at der bliver frigivet en del potentiel energi fra ballasten, og at denne til dels placeres som bevægelsesenergi i ballasten. For begge systemer gælder det, at umiddelbart inden det er optimalt at slippe projektilet, overføres en stor del af ballastens kinetiske energi til projektilet (det passer med at ballasten ikke kan falde længere).

For at finde forskellen mellem de to blider er det interessant at se, hvordan den kinetiske energi i projektilet bliver udnyttet til at kaste det fremad. Ud fra det skrå kast kan man opstille en sammenhæng mellem den tilgængelige energi, og den strækning et objekt kan kastes. På figur 12 og 13, er denne approksimering af kastelængden, sammenlignet med den energi systemet aktuelt opnår. For den optimerede blide (figur 12) ses det, at projektilets kinetiske energi bliver udnyttet optimalt omkring skudøjeblikket. Det samme er *ikke* tilfældet når slyngposen forlænges (figur 13), det ses at den tilgængelige energi udnyttes dårligt. Når man ser simuleringerne som animationer (Video 3 og 4), ses det, at projektilet i den optimerede blide har en vinkel i forhold til vandret på ca. 45% på det optimale skudtidspunkt, mens den anden blide først rammer denne vinkel, efter at ballasten har begyndt en sidelæns bevægelse og dermed har bundet noget energi.

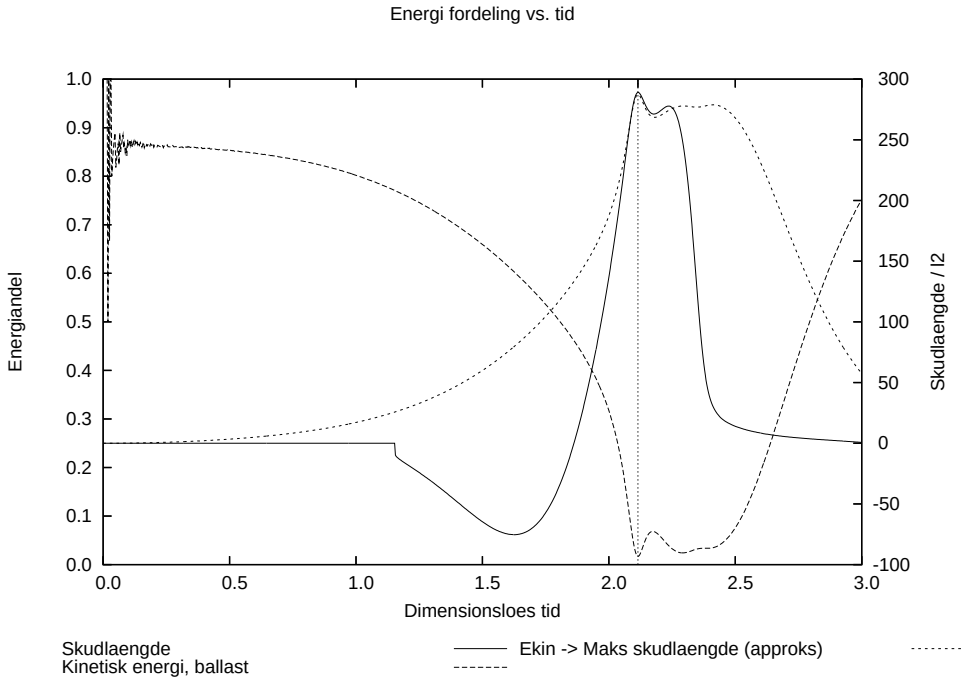
Det er muligt at estimere blidens virkningsgrad ud fra den opnåede skudlængde. Til det anvendes en *black box* model af bliden, der er i stand til at flytte al den potentielle energi i ballasten over i projektilet som kinetisk energi. Den



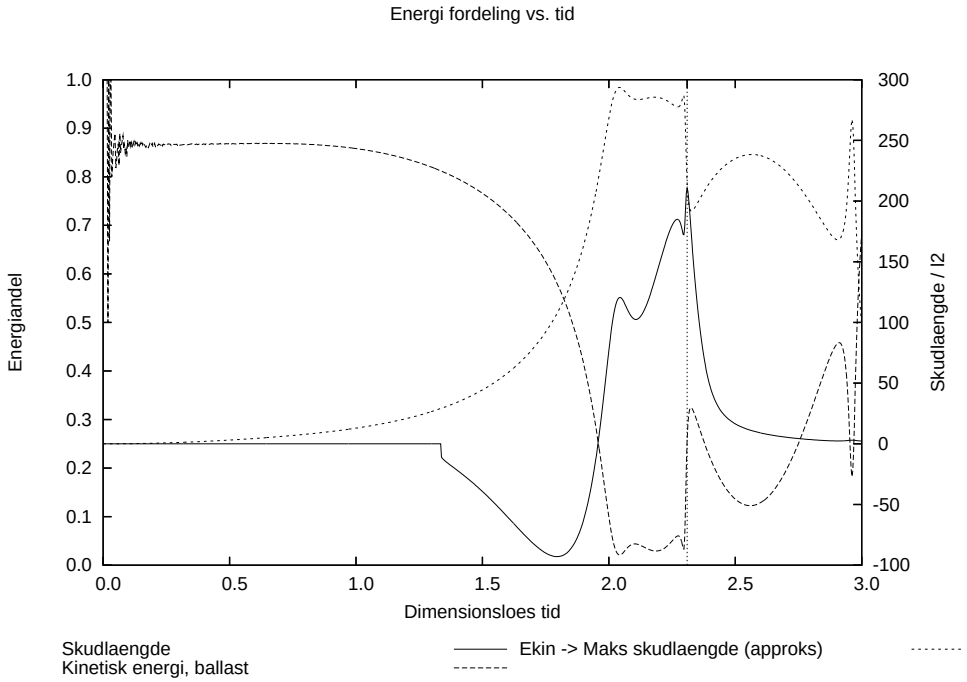
Figur 10: *Energi i de forskellige dele af systemet under affyringen af en optimeret blide. Bemærk hvor lidt kinetisk energi, der er bundet i ballasten, ved tidspunktet hvor projektilet bliver kastet længst (den markerede streg).*



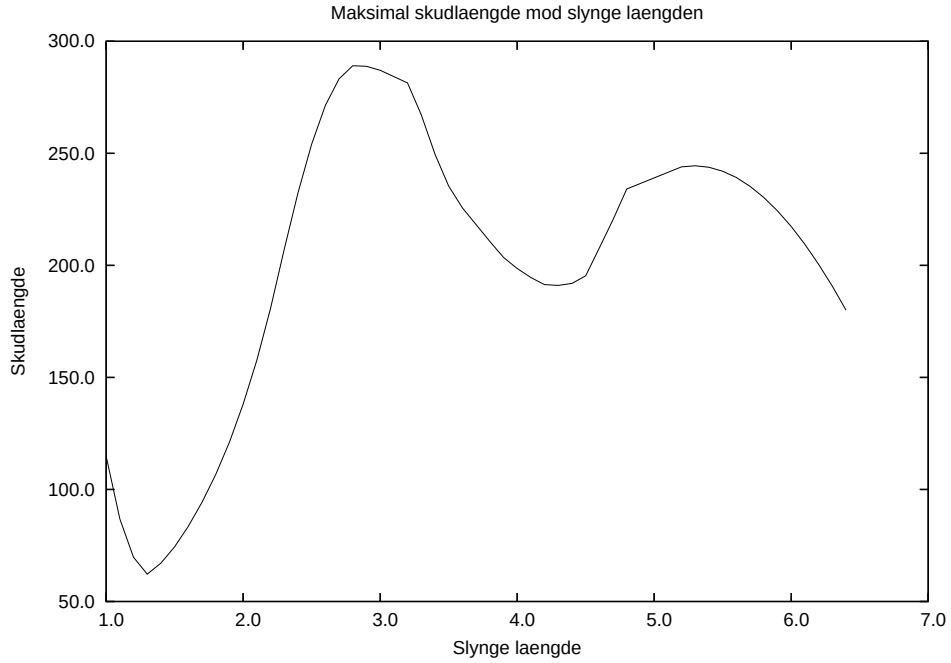
Figur 11: *Energi i de forskellige dele af systemet under affyringen af en blide med en for lang slyngpose. Modsat figur 10, er der her en større del energi bundet i ballastens bevægelse, omkring det optimale skudtidspunkt.*



Figur 12: *Lokalisering af energi under en affyring for en optimeret blide. Bemærk at ved det tidspunkt, hvor der vil blive kastet længst, er mindre end 2% af den frigivne potentielle energi bundet som kinetisk energi i ballasten.*



Figur 13: *Lokalisering af energi under en affyring for en blide med en for lang slyngpose. Bemærk at ved det tidspunkt, hvor der vil blive kastet længst, er omkrin 25% af den frigivne potentielle energi bundet som kinetisk energi i ballasten.*



Figur 14: *Maksimal skudlængde som funktion af slyngposens længde. Det andet lokale maksima omkring slyngelængde* $= 5.2l_2$*, er ikke reel. Det skyldes, at en af model antagelserne (at der altid trækkes i slyngposen) ikke holder, og simuleringen derfor laver en fejl.*

maksimalt opnåelige skudlængde, er så givet ved:

$$R_{\max} = 2 \cdot \frac{m_2}{m_1} \cdot h$$

hvor h er faldlængden for ballasten. For en standard blide er $R_{\max}$ givet ved:

$$R_{\max,\,\mathrm{std}} = 2 \cdot \frac{100kg}{1kg} \cdot \left(1 + \frac{1}{\sqrt{2}}\right) = 341.4m$$

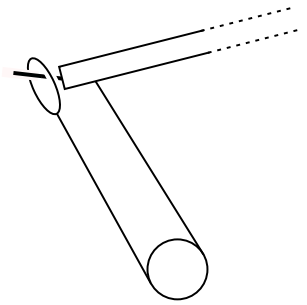
Virkningsgraderne kan så bestemmes til:

$$\frac{289.1m}{R_{\max,\,\mathrm{std}}} = 85\% \qquad \frac{210.7m}{R_{\max,\,\mathrm{std}}} = 62\%$$

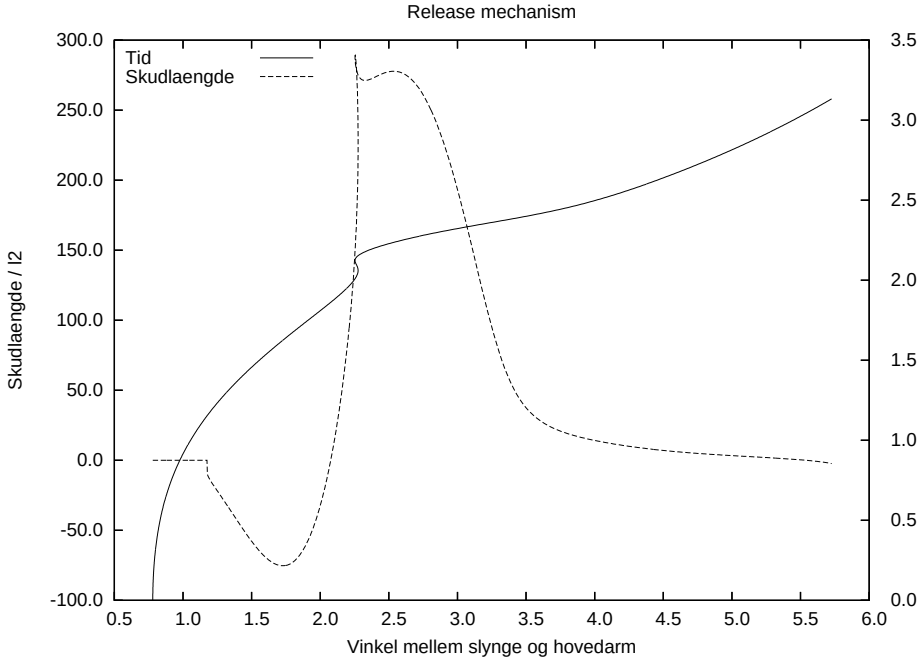
Tager man i betragtning hvor enkelt bliden er, er det en meget høj virkningsgrad.

Det skal nu undersøges hvordan længden af slyngposen påvirker den opnåelige skudlængde, på figur 14 er denne sammenhæng afbilledet. Det ses, at blidens virkningsgrad afhænger kraftigt af slyngposens længde, med et maksimum omkring $l_3 = 2.8l_2$. Maksimaet omkring $l_3 \sim 5.2l_2$ er ikke reelt, da skudlængden er opnået *efter* at modellen har udvist en ikke fysisk opførsel.

Det sidste jeg vil undersøge er frigørelses mekanismen, se en skitse af denne på figur 15. Projektilet bliver frigjort, når den ene ende af slyngposen (der er udformet som en løkke) glider af en krog monteret på hovedarmen. Krogen er udformet således, at vinklen mellem slyngposen og hovedarmen skal være over



Figur 15: *Skitse af frigørelsesmekanismen til slyngposen. Den ene ende af slyngposen er fastmonteret på hovedarmen, mens den anden ende, der er udformet som en løkke, hænger på en krog. I det projektilet svinges rundt under affyringen, glider løkken af krogen og projektilet frigøres.*



Figur 16: *Undersøgelse af frigørelses mekanismen. Bemærk plateauet omkring vinklen* $= 2.5Rad$*, det er her vi ønsket at slippe projektilet. Ved at benytte løkke og krog metoden (vist på figur 15), er det muligt at ramme en god affyringsvinkel hver gang.*

en tærskelværdi, inden løkken kan glide af. Jeg ønsker at undersøge hvor præcist krogen skal være vinklet, for at opnå en næsten optimal skudlængde. På figur 16, er sammenhængen mellem vinklen mellem slyngposen og hovedarmen afbilledet imod den aktuelle skudlængde. Det ses, at der er et plateau omkring en vinkel på $2.5Rad$, hvor bliden kaster mere end $250l_2$. Plateauets bredde svarer til ca. 30°, hvilket betyder, at det er let at indstille krogen så bliden skyder langt.

8 Konklusion

En simpel model af en blide med hængslet ballast er beskrevet. I modellen ses der bort fra friktion og elastiske kræfter.

Vha. dimensionsanalyse, er det undersøgt hvordan skudlængde og optimalt skudtidspunkt skalerer. Hvis alle dimensioner (længder) af bliden øges med en faktor γ:

- øges skudlængden med en faktor γ
- øges skudtiden med en faktor $\sqrt{\gamma}$

Der er fundet bevægelsesligninger for modellen vha. Lagrange formalismen.

Der er forsøgt at finde en analytisk løsning til bevægelsesligningerne vha. Hamilton–Jacobi metoden, det lykkes ikke, da det ikke er muligt at lave separation af variable på et bestemt udtryk. At Hamilton–Jacobi metoden fejler, udelukker ikke, at der findes en analytisk løsning til bevægelsesligningerne.

Til at sandsynliggøre at der ikke findes en analytisk løsning, benyttes et argument fra beregnelighedsteori. Resultatet er, at hvis der findes en analytisk løsning til blidens bevægelse, kan denne løsning reduceres til et kaotisk pendul, som sandsynligvis ikke har en analytisk løsning.

Der implementeres en numerisk løsning af de opstillede bevægelsesligninger. Den numeriske løsning er baseret på en fjerde ordens Runge–Kutta metode.

Rigtigheden af den implementerede løsning sandsynliggøres, ved at bestemme metodens orden ud fra de resultater som den numeriske løsning giver. Fejlen i den konstante energi konvergerer som $O(h^4)$, og fejlen på den maksimale skudlængde konvergerer som $O(h^2)$, hvilket i begge tilfælde er som forventet.

Til sidst er det undersøgt hvordan energien flyttes rundt i bliden under en affyring. Her findes det, at for en optimeret blide (med en masseløs hovedarm), er virkningsgraden 85%.

8.1 Perspektivering

Når man sætter sig grundigt ind i et bestemt emne, opdager man mange ting, der også kunne være interessante at undersøge; lige præcis det har jeg også oplevet i forbindelse med dette projekt. Nogle af de ting er gengivet herunder:

- Undersøg hvad forholdet mellem vægten af projektilet og ballasten betyder for blidens funktion.
- Undersøg hvor stor en mekanisk belastning de enkelte dele af bliden udsættes for.
- Udvid simuleringen til også at tage højde for
  - Masse i hovedarmen
  - Sliske, der holder projektilet oppe under den første del af affyringen
- Find proportionerne af en optimeret blide. Dette problem kan angribes på flere måder, f.eks. vha. en genetisk algoritme.
- Energi overførsel i svingende systemer (f.eks. mellem to koblede penduler). Resultater herfra kan måske give mere indsigt i blidens virkemåde. Se beskrivelse af et simpelt eksperiment her
  `http://fysikbasen.dk/index.php?page=Vis&id=82`
- Moderne forbedringer af bliden, herunder: "Floating Arm Trebuchet", "scissor-jack" og "Arm Slides Over Cam", se mere her
  `http://www.trebuchet.com/story.php/fat.html`
  `http://www.siege-engine.com/ScissorTreb.shtml`
  `http://www.siege-engine.com/BabyASOK.shtml`

A Programmet

Det udviklede program til simulering af en hængslet blide, er lagt på nettet til fri afbenyttelse, under en såkaldt MIT licens. Denne licens betyder at alle frit kan benytte programmet og arbejde videre på koden, sålænge at koden stadig indeholder (og dermed er underlagt) MIT licensen og udviklerens navn.

A.1 Krav til systemet

For at den rå simulering kan køre skal der være en *python fortolker* til rådighed. Ønsker man at se simuleringen som en animation, er en fungerende installation af gnuplot, samt gnuplot.py[9] nødvendig. Programmerne kan hentes fra:

- `http://www.python.org/`
- `http://www.gnuplot.info/`
- `http://gnuplot-py.sourceforge.net/`

Selve simulatoren kan hentes fra `http://www.fys.sdu.dk/fysikshow/blide/`.

A.2 Hvordan bruges programmet?

Programmet køres via kommando linien, og tager en række parametre. Køres programmet uden ekstra parametre, startes simuleringen med en "standard" blide[10] med en skridtlængde på $h = 0.00001$.

```
% python blide.py
   1.000  100.000    4.000    1.000    4.000    1.000
        0.780    0.000   -1.570   0.22105 176.84122496297      1e-05
```

Den linie med resultater som programmet udskriver, indeholder følgende.

m_1 m_2 l_1 l_2 l_3 l_4 θ_1 θ_2 θ_3 virkningsgrad skudlængde skridtlængde

hvor "virkningsgrad" er defineret ved forholdet mellem den opnåede skudlængde og det maksimalt opnåelige.

Er man interesseret i at undersøge en bestemt konfiguration / start stilling mht. hvor langt bliden kan skyde, gøres det med følgende parametre:

```
% python blide.py --position 1 100 1 1 1 1 0.78 0.0 -1.57
   1.000  100.000    1.000    1.000    1.000    1.000
        0.780    0.000   -1.570   0.26296 210.36472618671      1e-05
```

[9] python interface til gnuplot

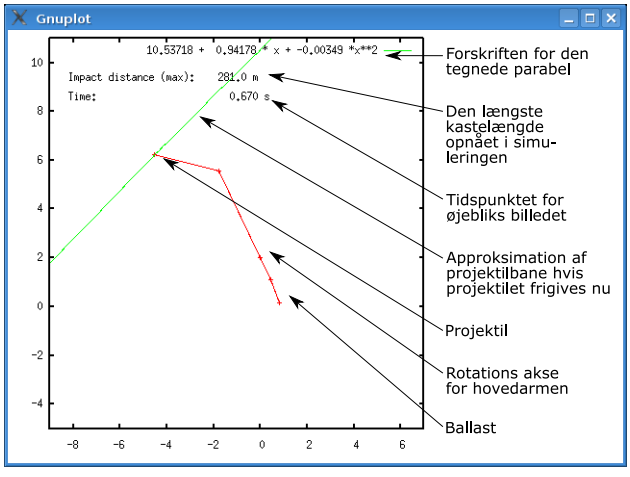
[10] Dimensioner på en standard blide: $m_1 = 1$, $m_2 = 100$, $l_1 = 4$, $l_2 = 1$, $l_3 = 4$, $l_4 = 1$, $\theta_1 = 0.78$, $\theta_2 = 0.00$ og $\theta_3 = -1.57$.

Programmet kan vise hvilke kommandolinie parametre det tager, ved at kalde det på denne måde:

```
% python blide.py --help
usage: blide.py [options]

options:
  -h, --help            show this help message and exit
  --gnuplot             Toggle gnuplot animations
  --energies            Show info on location of kinetic and potential energy
  --position=m1 m2 l1 l2 l3 l4 t1 t2 t3
                        Set initial position of the trebuchet. m1: mass of
                        projectile, m2: mass of counterweight, l1: length fro
                        pivot to sling attach, l2: length of xx, l3: length o
                        xx, l4: length of xx, t1: angle of main arm (in
                        radians), t2: angle of xxx (in radians), t3: angle of
                        xxx (in radians)
  --angles              Show coordinate values in each iteration.
  --coordinates         Show coordinate values in each iteration.
  --impact              Show info on impact location.
  --redrawtime=REDRAWTIME
                        Set time between redraws in the animation.
  --steplength=STEPLENGTH
                        Set steplength in seconds.
  --gravitationalacceleration=G
                        Set gravitational acceleration.
  --nosummary           Display summary (initial position and found throwing
                        length).
  --reductionfactor=factor
                        Reduce amount of output by factor.
```

På figur 17 er vist et billede fra simuleringen, med forklaringer til hvad figuren viser.



Figur 17: *Skærmbillede fra animationen, med forklaringer til de viste elementer.*

B Uddybende forklaring af figurer

Ofte er der ikke plads til at beskrive medtagne figurer i detaljer, derfor dette bilag, hvor der er plads til længere forklaringer / flere detajler i beskrivelsen af en figur. I bilaget er der en oversigt over de enkelte figurer med kommentarer til, hvordan de er lavet, hvad de viser, og hvilke parametre jeg har benyttet til eventuelle simuleringer.

B.1 Billede fra en simulering, figur 6 side 17.

Billedet er et øjebliksbillede fra simulering, omkring et tidspunkt hvor simuleringen ikke stemmer overens med virkeligheden.

B.2 Konvergens for den numeriske metode, figur 7 side 18.

Figuren viser sammenhængen mellem den anvendte skridtlængde i simuleringen og den fundne kastelængde. Meningen med figuren var, grafisk, at checke ordenen af den anvendte metode, ved at se hvilken potens–funktion som fejlen gik efter. Det giver problemer, da kurven er meget svingende ved store skridtlængder, og det giver ikke mening at fitte en kurve til denne region.

I stedet har jeg tegnet kurver for en fejl der går som $O(h)$, $O(h^2)$ hhv. $O(h^3)$, de tre kurver er tilpasset sådan at de går mod 289.1 når $h \to 0$ og, at de går gennem punktet $(0.0116; 288.384)$, der er en af de plottede punkter.

Det ses, at kurven for $O(h^2)$ virker som en indhylnings–kurve for de fundne resultater, den viser, at fejlen på den bestemte skudlængde er begrænset af $O(h^2)$.

Den fundne orden stemmer godt overens med den måde værdien bliver bestemt på. Se på skudlængden som en kontinuert funktion af tiden, vi er interesserede i at finde den maksimale værdi af denne funktion. Det gøres ved at beregne funktionsværdien i diskrete punkter med en indbyrdes afstand på h. Ved at vælge tiden for den største beregnede funktionsværdi (t_0), ved vi nu, at det rigtige maksimum ligger i intervallet $[t_0 - h/2; t_0 + h/2]$, altså fejlen er af samme størrelsesorden som h. Et maksimum af en kontinuert funktion kan approksimeres ved en parabel, centreret omkring maksimaet. Højde forskellen fra maksimaet er så proportional med afstanden fra maksimaet (h) i anden, dvs. $O(h^2)$.

Data til kurven er lavet med kommandoer i stil med følgende:

```
python blide.py --position 1 100 4 1 2.8 1 .78 .0 -1.57
    --steplength 0.0001
```

B.3 Skridtlængde og energi bevarelse, figur 8 side 18.

Figuren viser hvordan systemets samlede energi ændrer sig i forhold til start energien (reelt er det den absolutte ændring af energien, der er afbilledet, det er gjort for at gøre det muligt at benytte en logaritmisk skala).

For en analytisk løsning til bevægelsesligningerne vil kurven være nul, men da en numerisk løsning kun kan være en approksimation, må det forventes, at der er mindre ændringer i den samlede energi.

Det, at den samlede energi ændres er en fejl i det aktuelle system, og denne ændring kan være et mål for hvor stor en fejl, der er i hele simuleringen.

Der er kørt fire simuleringer med forskellige skridtlængder, og på figuren er ændringen i den samlede energi afbilledet i forhold til tidspunktet i simuleringen. Det ses, umiddelbart, at de fire kurver minder rigtig meget om hinanden, der er næsten kun en faktor 16 til forskel mellem to nabo–kurver.

Det bemærkes, at fejlen ikke vokser jævnt med tiden, men at den primært øges i bestemte perioder. Det er denne observation, der ligger bag ideen om at løse sådanne problemer med en variabel skridtlængde.

Data til kurven er lavet med kommandoer med følgende opbygning:

```
python blide.py --position 1 100 4 1 2.8 1 .78 .0 -1.57
--steplength h --energies --nosummary
```

Hvor h udskiftes med den aktuelle skridtlængde.

B.4 Vinkler over tid, figur 9 side 20.

Figuren viser hvordan de forskellige vinkler udvikles over tid. Der er en interessant overgang i figuren. Dette er tidspunktet hvor hældningen af hovedarmen og ballastarmen er ens. Inden dette tidspunkt er tyngdens træk i ballasten med til at accelerere hovedarmen i en roterende bevægelse. Efter tidspunktet begynder hovedarmen at sætte ballasten i en sidelæns bevægelse, hvorved der bindes energi i ballasten og det må forventes at skudlængden vil mindskes. Det markerede tidspunkt, der svarer til det optimale skudtidspunkt, ligger samtidig som den beskrevne overgang.

Data til figuren er lavet med kommandoen:

```
python blide.py --position 1 100 4 1 2.8 1 .78 .0 -1.57
    --steplength 0.0001 --angles --nosummary
```

B.5 Energi lokalisering, figur 10 og figur 11 side 22.

Figurerne viser hvor i systemet energien er lokaliseret til et givet tidspunkt. Ser man på ballastens kinetiske energi, er det tydeligt at ballasten bremses kraftigt op lige inden det optimale skudtidspunkt.

B.6 Energi udnyttelse, figur 12 og figur 13 side 23.

Figurerne viser tre grafer: (1) Andelen af den frigivne potentielle energi, der er bundet som kinetisk energi i ballasten, (2) skudlængde som funktion af tiden. Værdierne bliver kun beregnet når projektilbanen vil skære x-aksen og (3) et estimat af en øvre grænse for hvor langt projektilet kan kastes, hvis det sendes af sted fra jordoverfladen, i den rigtige retning, med den aktuelle kinetiske energi. Til (2) og (3) ses der bort fra luftmodstand, da projektilbanerne antages at være parabelformede.

Figur 12 viser at al den tilgængelige energi udnyttes på det optimale skudtidspunkt til at sende projektilet af sted. Det ses at den opnåede skudlængde

overstiger estimatet af hvor langt det er muligt at kaste projektilet, det kan forklares ved at projektilet på skudtidspunktet er hævet et stykke over jordoverfladen. Den optimerede blide overfører altså næsten al energien til projektilet, for derefter at slippe det i en (næsten) optimal vinkel. På skudtidspunktet er 2% af den frigivne energi lokaliseret som kinetisk energi i ballasten, mens næsten al den resterende energi kan findes i projektilet.

Figur 13 viser, at den opnåede skudlængde er væsentligt mindre end på den forrige figur. Det skyldes, at projektilet er bagud i bevægelsen, da en optimal skudlængde først opnås efter, det tidspunkt hvor, ballasten er begyndt at bremse hovedarmen. Den ikke optimerede blide er i stand til at flytte hovedparten af den frigivne energi over i projektilet, men er ikke i stand til at udnytte denne mængde energi optimalt.

Data til figur 12 er lavet med kommandoen:

```
python blide.py --position 1 100 4 1 2.8 1 .78 .0 -1.57
    --steplength 0.001 --energies --angles --impact --nosummary
```

Data til figur 13 er lavet med kommandoen:

```
python blide.py --position 1 100 4 1 3.8 1 .78 .0 -1.57
    --steplength 0.001 --energies --angles --impact --nosummary
```

B.7 Skudlængde og slyngposens længde figur 14 side 24.

Figuren afbilleder sammenhængen mellem slyngposens længde og den maksimale skudlængde. Det ses, at blidens virkningsgrad afhænger kraftigt af slyngposens længde, med et maksimum omkring $l_3 = 2.8l_2$. Maksimaet omkring $l_3 \sim 5.2l_2$ er ikke reelt, da skudlængden er opnået *efter*, at slyngposen burde være krøllet sammen (forkert antagelse under modelleringen, der i enkelte tilfælde giver modellen en forkert opførsel.).

Data til figuren er lavet med kommandoer af typen:

```
python blide.py --position 1 100 4 1 l3 1 .78 .0 -1.57
    --steplength 0.001 --energies --angles --impact --nosummary
```

Hvor l3 er udskiftet med den skudlængde der nu prøves med.

B.8 Frigørelses mekanismen, figur 16 side 25.

På figuren er skudlængden afbilledet mod vinklen mellem hovedarmen og slyngposen. Det ses, at skudlængden har et højt niveau i intervallet $[2.2 : 2.7]$, hvilket svarer til et spænd på næsten 30°. Det er altså let, at indstille bliden til at skyde langt, da slyngen har "lang" tid til at slippe krogen.

Data til figuren er lavet med kommandoen:

```
python blide.py --position 1 100 4 1 2.8 1 .78 .0 -1.57
    --steplength 0.001 --energies --angles --impact --nosummary
```

C Richardson ekstrapolation

Richardson ekstrapolation [Christiansen(1999), side 102] kan benyttes, når man skal undersøge resultatet af en numerisk metode, der afhænger af en variabel skridtlængde. Metoden antager, at resultatet af den numeriske metode kan skrives som en funktion af skridtlængden. Denne funktion taylorudvikles omkring $x = 0$, og man har så (Alle eksponenterne er større end nul: $\alpha_i > 0$):

$$A(x) \approx a_0 + a_1 x^{\alpha_1} + a_2 x^{\alpha_2} + a_5 x^{\alpha_3} + \ldots$$

Det, vi egentligt er interesserede i, er værdien af $A(x)$, når $x \to 0$, dvs. værdien a_0. Metodens orden er den mindste værdi af α_i. Ser man bort fra de højere ordens led (der for små h værdier er forsvindende), kan følgende relation mellem resultatet fra tre forskellige skridtlængder (h, βh og $\beta^2 h$) og metodens orden α_1 bestemmes:

$$\frac{A(\beta^2 h) - A(\beta h)}{A(h) - A(\beta h)} \simeq \frac{(\beta^2 h)^{\alpha_1} - (\beta h)^{\alpha_1}}{(\beta h)^{\alpha_1} - h^{\alpha_1}} = \frac{\beta^{2\alpha_1} - \beta^{\alpha_1}}{\beta^{\alpha_1} - 1} = \beta^{\alpha_1} \tag{C.1}$$

Givet to resultater opnået med forskellige skridtlængder og viden om en metodes orden, er det muligt at fjerne det mest betydende fejlled (a_1) og derved forbedre resultatet ganske væsentligt. For at det er muligt at ekstrapolere, skal fejlen fra den laveste orden være dominerende (dvs. at $|a_1 h^{\alpha_1}| >> |a_2 h^{\alpha_2}|$), hvilket kan opnås ved at benytte tilstrækkeligt små skridtlængder. Der ekstrapoleres ved at beregne værdien af følgende udtryk:

$$\begin{aligned} A(h) - \frac{A(\beta h) - A(h)}{\beta^{\alpha_1} - 1} &= a_0 + a_1 h^{\alpha_1} - \frac{a_1(\beta h)^{\alpha_1} - a_1 h^{\alpha_1}}{\beta^{\alpha_1} - 1} - \frac{a_2(\beta h)^{\alpha_2} - a_2 h^{\alpha_2}}{\beta^{\alpha_1} - 1} \\ &= a_0 + a_1 h^{\alpha_1} - a_1 h^{\alpha_1} \frac{\beta^{\alpha_1} - 1}{\beta^{\alpha_1} - 1} - a_2 h^{\alpha_2} \frac{\beta^{\alpha_2} - 1}{\beta^{\alpha_1} - 1} \\ &= a_0 - a_2 h^{\alpha_2} \frac{\beta^{\alpha_2} - 1}{\beta^{\alpha_1} - 1} \end{aligned} \tag{C.2}$$

Kender man værdierne $A(\beta h)$ og $A(h)$, samt metodens orden α_1, er ekstrapolationen til $A(0)$ givet ved:

$$A(0) \simeq A(h) - \frac{A(\beta h) - A(h)}{\beta^{\alpha_1} - 1} \tag{C.3}$$

Ideen med Richardson ekstrapolation er, at man kan fjerne leddet med den mindste eksponent, ved at sammenligne nogle resultater med forskellige skridtlængder. Når fejlleddet med den mindste eksponent fjernes, øges metodens orden til den næstmindste værdi af α_i.

D Hamilton–Jacobi ligningen i en nøddeskal

D.1 Ligning

Hamilton–funktionen for det mekaniske system betegnes:

$$H = H(q_1, \ldots, q_s, p_1, \ldots, p_s, t)$$

Virkningsfunktionen (action) betegnes:

$$S = S(q_1, \ldots, q_s, t)$$

Der gælder:

$$\frac{\partial S}{\partial t} = -H, \qquad \frac{\partial S}{\partial q_i} = p_i, \qquad i = 1, \ldots, s$$

Heraf fås straks Hamilton–Jacobi (HJ) ligningen:

$$\frac{\partial S}{\partial t}(q_1, \ldots, q_s, t) + H(q_1, \ldots, q_s, \frac{\partial S}{\partial q_1}, \ldots, \frac{\partial S}{\partial q_s}, t) = 0$$

D.2 Hamilton–Jacobi metoden

Find en løsning til HJ ligningen på formen:

$$S = F(q_1, \ldots, q_s, t, \alpha_1, \ldots, \alpha_s) + A \tag{D.1}$$

hvor $\alpha_1, \ldots, \alpha_s$ og A er arbitrære konstanter. Ved brug af kanoniske transformationer vises det at:

$$\frac{\partial S}{\partial \alpha_1} \qquad \frac{\partial S}{\partial \alpha_2} \qquad \cdots \qquad \frac{\partial S}{\partial \alpha_s}$$

er bevarede størrelser. Dvs. de s ligninger, $\frac{\partial S}{\partial \alpha_i} = \beta_i$, hvor β_i sammen med α_i er arbitrære konstanter, giver den generelle løsning til bevægelsesproblemet.

D.3 Separation af variable

I mange vigtige tilfælde kan man finde en løsning (D.1) med følgende metode:

Antag at en af koordinaterne, f.eks. q_1, kun optræder i HJ ligningen i en kombination $f\left(q_1, \frac{\partial S}{\partial q_1}\right)$ der ikke indolverer tiden t, de andre koordinater $q_2, \ldots, q_s$ samt de afledede $\frac{\partial S}{\partial q_2}, \ldots \frac{\partial S}{\partial q_s}$. Dvs. HJ ligningen har den generelle form:

$$G\left(q_2, \ldots, q_s, \frac{\partial S}{\partial q_2}, \ldots, \frac{\partial S}{\partial q_s}, t, f\left(q_1, \frac{\partial S}{\partial q_1}\right)\right) = 0 \tag{D.2}$$

hvor G er en eller anden funktion.

Vi kan nu straks indføre en arbitrær konstant α_1 ved:

$$f\left(q_1, \frac{\partial S}{\partial q_1}\right) = \alpha_1 \tag{D.3}$$

og antage en løsning til HJ ligningen på formen:

$$S(q_1, \ldots, q_s, t) = S_1(q_1) + S'(q_2, \ldots, q_s, t)$$

Indsættes dette i (D.2) fås

$$G\left(q_2, \ldots, q_s, \frac{\partial S'}{\partial q_2}, \ldots, \frac{\partial S'}{\partial q_s}, t, \alpha_1\right) = 0$$

Dvs. en "ny" HJ ligning med en variabel mindre og hvori der indgår den arbitrære konstant α_1.

Hvis vi kan fortsætte på denne måde kan en løsning på formel (D.1) findes og bevægelsesproblemet løses generelt. Bemærk at (D.3) er en bevarelses lov.

D.4 Cykliske variable

Antag at en generaliseret koordinat, f.eks. q_1, ikke optræder eksplicit i Hamilton–funktionen, dvs.:

$$H = H(q_2, \ldots, q_s, p_1, \ldots, p_s, t)$$

HJ ligningen får her formen:

$$\frac{\partial S}{\partial t} + H\left(q_2, \ldots, q_s, \frac{\partial S}{\partial q_1}, \ldots, \frac{\partial S}{\partial q_s}, t\right) = 0$$

Her reducerer funktionen $f\left(q_1, \frac{\partial S}{\partial q_1}\right)$ til $\frac{\partial S}{\partial q_1}$. Vi skriver derfor:

$$\frac{\partial S}{\partial q_1} = \alpha_1, \qquad S = S_1(q_1) + S'(q_2, \ldots, q_s, t)$$

Som straks giver:

$$\frac{\partial S}{\partial q_1} = \frac{dS_1}{dq_1} = \alpha_1$$

og dette medfører:

$$S_1(q_1) = \alpha_1 \cdot q_1$$

samt den "nye" HJ ligning for funktionen $S'(q_2, \ldots, q_s, t)$:

$$\frac{\partial S'}{\partial t} + H\left(q_2, \ldots, q_s, \frac{\partial S'}{\partial q_2}, \ldots, \frac{\partial S'}{\partial q_s}, t\right) = 0$$

Bemærk at $\frac{\partial S}{\partial q_1} = p_1$, så $p_1 = \alpha_1$ er bevaret og S har formen:

$$S = p_1 \cdot q_1 + S'(q_2, \ldots, q_s, t)$$

Et vigtigt special tilfælde forekommer når tiden t ikke indgår eksplicit i Hamilton funktionen. Her kan vi straks skrive:

$$\frac{\partial S}{\partial t} = -H = -E, \qquad S = S_t(t) + S'(q_1, \ldots, q_s)$$

hvor E er en arbitrær konstant (energien). Dette giver straks:

$$\frac{\partial S}{\partial t} = \frac{dS_t}{dt} = -E \qquad\qquad S_t(t) = -E \cdot t$$

samt den "nye" HJ ligning for funktionen $S'(q_1, \ldots, q_s)$:

$$-E + H\left(q_1, \ldots, q_2, \frac{\partial S}{\partial q_1}, \ldots, \frac{\partial S}{\partial q_s}\right) = 0$$

E Programkoden

```
#! /usr/bin/python
# This Python file uses the following encoding: utf-8

# Author:         Henrik Skov Midtiby
# Institution:    University of Southern Denmark
# Last changed: 2006-05-31

'''
The MIT License

Copyright (c) 2006 Henrik Skov Midtiby

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation files
(the "Software"), to deal in the Software without restriction,
including without limitation the rights to use, copy, modify, merge,
publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
'''

# Class which can simulate a hinged counterweight trebuchet.
class Blide:
    def __init__(self):
        # Define standard proportions of the trebuchet.
        # Lengths (in meters)
        self.l1 = 4
        self.l2 = 1
        self.l3 = 4
        self.l4 = 1

        # Masses (in kilograms)
        # Projectile
        self.m1 = 1
        # Ballast
        self.m2 = 100

        # Default starting conditions for the simulation
        # Generalized coordinates
        self.theta1c =  0.78
        self.theta2c =  0.00
        self.theta3c = -1.57

        # Generalized velocities
        self.theta1v = 0.0
        self.theta2v = 0.0
        self.theta3v = 0.0

        # Generalized accelerations
        self.theta1a = 0.0
        self.theta2a = 0.0
        self.theta3a = 0.0

        # Time step length
        self.timestep = timestepInit
        self.time = 0
        self.maxLength = 0
        self.maxLengthTeoretic = 4* self.m2 / self.m1 \
                * (self.l2 + self.l4 )

        # Various
        self.lastTimePlottet = 0
        self.counter = 0
        self.epotprojectileref = "not defined"
        self.epotcounterweightref = "not defined"

    # Calculate the "derivative" of the partial diffential
    # equations (velocities and accelerations).
    def calcAccelerations(self, m1, m2, l1, l2, l3, l4, t1c, t2c, t3c, t1v, t2v, t3v):
        # This is pure magic found by using the lagrange formalism
        # It is explained in the paper
        vala = m1 * l1**2 + m2 * l2**2
        valb = - m1 * l1 * l3 * math.cos(t1c - t2c)
        valc = m2 * l2 * l4 * math.cos(t1c - t3c)
        vald = g * math.cos(t1c) * (m1 * l1 - m2 * l2) \
```

```
                + m1 * l1 * l3 * t2v**2 * math.sin(t1c - t2c) \
89              - m2 * l2 * l4 * t3v**2 * math.sin(t1c - t3c)
        vale = - l1 * math.cos(t1c - t2c)
91      valf = l3
        valg = - g * math.cos(t2c) \
93              - l1 * t1v**2 * math.sin(t1c - t2c)
        valh = l2 * math.cos(t1c - t3c)
95      vali = l4
        valj = - g * math.cos(t3c) \
97              + l2 * t1v**2 * math.sin(t1c - t3c)

99      # Solved three equations with three unknown.
        t1a = (-valb * valg * vali - valc * valf * valj + vald * valf * vali) \
101                 / (vala * valf * vali - valb * vale * vali - valc * valf * valh)
        t2a = (valb*vale*valg*vali + valc*vale*valf*valj - vald*vale*valf*vali) \
103                 / (vala*valf**2*vali - valb*vale*valf*vali - valc*valf**2*valh) \
                    + valg / valf
105     t3a = (vala*valf*valj - valb*vale*valj + valb*valg*valh - vald*valf*valh) \
                    / (vala*valf*vali - valb*vale*vali - valc*valf*valh)
107
        return (t1v, t2v, t3v, t1a, t2a, t3a)
109
    # Take one step with the fourth order Runge-Kutta method.
111 # The method is described on wikipedia
    # http://en.wikipedia.org/wiki/Runge_kutta
113 def updateVelocitiesAndPositions(self):
        # Calculate temporary values
115     ts = self.timestep

117     k1 = self.calcAccelerations(
                self.m1, self.m2, self.l1, self.l2, self.l3, self.l4,
119             self.theta1c,
                self.theta2c,
121             self.theta3c,
                self.theta1v,
123             self.theta2v,
                self.theta3v)
125
        k2 = self.calcAccelerations(
127             self.m1, self.m2, self.l1, self.l2, self.l3, self.l4,
                self.theta1c + 0.5 * ts * k1[0],
129             self.theta2c + 0.5 * ts * k1[1],
                self.theta3c + 0.5 * ts * k1[2],
131             self.theta1v + 0.5 * ts * k1[3],
                self.theta2v + 0.5 * ts * k1[4],
133             self.theta3v + 0.5 * ts * k1[5])

135     k3 = self.calcAccelerations(
                self.m1, self.m2, self.l1, self.l2, self.l3, self.l4,
137             self.theta1c + 0.5 * ts * k2[0],
                self.theta2c + 0.5 * ts * k2[1],
139             self.theta3c + 0.5 * ts * k2[2],
                self.theta1v + 0.5 * ts * k2[3],
141             self.theta2v + 0.5 * ts * k2[4],
                self.theta3v + 0.5 * ts * k2[5])
143
        k4 = self.calcAccelerations(
145             self.m1, self.m2, self.l1, self.l2, self.l3, self.l4,
                self.theta1c + ts * k3[0],
147             self.theta2c + ts * k3[1],
                self.theta3c + ts * k3[2],
149             self.theta1v + ts * k3[3],
                self.theta2v + ts * k3[4],
151             self.theta3v + ts * k3[5])

153     # Update coordinates, velocities and the systems time.
        self.theta1c = self.theta1c + ts / 6 * (k1[0] + 2*k2[0] + 2*k3[0] + k4[0])
155     self.theta2c = self.theta2c + ts / 6 * (k1[1] + 2*k2[1] + 2*k3[1] + k4[1])
        self.theta3c = self.theta3c + ts / 6 * (k1[2] + 2*k2[2] + 2*k3[2] + k4[2])
157     self.theta1v = self.theta1v + ts / 6 * (k1[3] + 2*k2[3] + 2*k3[3] + k4[3])
        self.theta2v = self.theta2v + ts / 6 * (k1[4] + 2*k2[4] + 2*k3[4] + k4[4])
159     self.theta3v = self.theta3v + ts / 6 * (k1[5] + 2*k2[5] + 2*k3[5] + k4[5])
        self.time = self.time + ts
161
    # Iterate one step
163 def step(self):
        self.updateVelocitiesAndPositions()
165
    # Print various information as requested from the commandline.
167 # If the animation is requested update the plot if appropiate.
    def printPosition(self):
169     self.lastTimePlottet

171     # Calculate coordinates of specific parts of the trebuchet.
        # The end of the main beam connected to the projectile.
173     # The y-coordinate is raised by l2 + l4.
        pos2x = -self.l1 * math.cos(self.theta1c)
175     pos2y = -self.l1 * math.sin(self.theta1c) + self.l2 + self.l4
        # The end of the main beam connected to the counterweight.
177     # The y-coordinate is raised by l2 + l4.
```

```
            pos4x = self.l2 * math.cos(self.theta1c)
179         pos4y = self.l2 * math.sin(self.theta1c) + self.l2 + self.l4
            # Location of the projectile.
181         pos1x = pos2x + self.l3 * math.cos(self.theta2c)
            pos1y = pos2y + self.l3 * math.sin(self.theta2c)
183         # Location of the counterweight.
            pos5x = pos4x + self.l4 * math.cos(self.theta3c)
185         pos5y = pos4y + self.l4 * math.sin(self.theta3c)

187         # Calculate velocity vector of the projectile.
            vel1x = self.l1 * self.theta1v * math.sin(self.theta1c) \
189                 - self.l3 * self.theta2v * math.sin(self.theta2c)
            vel1y = - self.l1 * self.theta1v * math.cos(self.theta1c) \
191                 + self.l3 * self.theta2v * math.cos(self.theta2c)

193         # Calculate the potential energies in the system.
            epotprojectile = - self.m1 * self.l1 * g * math.sin(self.theta1c) \
195                 + g * self.m1 * self.l3 * math.sin(self.theta2c)
            epotcounterweight = self.m2 * self.l2 * g * math.sin(self.theta1c) \
197                 + g * self.m2 * self.l4 * math.sin(self.theta3c)

199         # Remember the first values of the potential energies.
            if self.epotprojectileref == "not defined":
201             self.epotprojectileref = epotprojectile
                self.epotcounterweightref = epotcounterweight
203
            # Change in potential energi of the current state of the system,
205         # compared to the initial state.
            epotprojectile    = epotprojectile - self.epotprojectileref
207         epotcounterweight = epotcounterweight - self.epotcounterweightref

209         # Kinetic energy of the projectile
            ekinprojectile = 0.5 * self.m1 * (self.l1**2 * self.theta1v**2  \
211                 + self.l3**2 * self.theta2v**2 - 2 * self.l1 * self.l3 \
                    * self.theta1v * self.theta2v * math.cos(self.theta1c - self.theta2c))
213         # Kinetic energy of the counterweight
            ekincounterweight = 0.5 * self.m2 * (self.l2**2 * self.theta1v**2 \
215                 + self.l4**2 * self.theta3v**2 + 2 * self.l2 * self.l4 \
                    * self.theta1v * self.theta3v * math.cos(self.theta1c - self.theta3c))
217         # The total mechanical energy.
            totalenergy = epotprojectile + epotcounterweight \
219                 + ekinprojectile + ekincounterweight

221         # Calculate impact location, by asuming that the projectile
            # is released in this moment and neglection friction.
223         # Calculated by solving a quadratic equation.
            if vel1y**2 + 2 * g * pos1y > 0:
225             impactx = pos1x + vel1x * vel1y / g \
                    + vel1x * math.sqrt(vel1y**2 + 2 * g * pos1y) / g
227             # If necessarily update the maximum impact range.
                if impactx > self.maxLength:
229                 self.maxLength = impactx
            else:
231             impactx = 0

233         # Update the animation if appropriate
            if self.time - self.lastTimePlottet > redrawTime and toggleGnuplot:
235             self.lastTimePlottet = self.time
                # Draw the actual position of the trebuchet
237             # with projectile, beam and ballast.
                actualPositionOfTheTrebuchet = [ [pos1x, pos1y], [pos2x, pos2y],
239                     [0, self.l2 + self.l4], [pos4x, pos4y], [pos5x, pos5y]]
                p1 = Gnuplot.Data(actualPositionOfTheTrebuchet)
241
                # Determine flightpath of projectile, if the projectile was free
243             # The path is approximated by a parabola.
                aa = - 0.5 * g / vel1x**2
245             bb = vel1y / vel1x + g * pos1x / vel1x**2
                cc = pos1y - vel1y / vel1x * pos1x - 0.5 * g * pos1x**2 / vel1x**2
247             p2 = Gnuplot.Func("%8.5f + %8.5f * x + %8.5f * x**2" % (cc, bb, aa))

249             # Set labels
                gp('unset label')
251             gp('set label "Impact distance (max): %8.1f m" at graph 0.05, 0.90' \
                    % self.maxLength)
253             gp('set label "Time: %8.3f s" at graph 0.05, 0.85' \
                        % self.time)
255             # Update the animation
                gp.plot(p1,p2)
257
            # In case something should be written to the screen.
259         if self.counter % divider == 0:
                if toggleEnergies or toggleAngles or toggleCoordinates:
261                 print "%8.5f" % (self.time),

263             if toggleAngles:
                    print "%8.5f %8.5f %8.5f" \
265                     % (self.theta1c, self.theta2c, self.theta3c),

267             if toggleCoordinates:
```

```
                print "%8.5f %8.5f %8.5f %8.5f %8.5f %8.5f %8.5f %8.5f" \
269                 % (pos2x, pos2y, pos1x, pos1y, pos4x, pos4y, pos5x, pos5y),

271             if toggleEnergies:
                    print "%8.1f %8.1f %8.1f %8.1f %14.10f" % (epotprojectile,
273                         epotcounterweight, ekinprojectile, ekincounterweight, totalenergy),

275             if toggleImpact:
                    print "%8.5f" % (impactx),
277
                if toggleEnergies or toggleAngles or toggleCoordinates:
279                 print
            self.counter += 1
281
        # Return a string with the initial conditions of the trebuchet.
283     def parameterString(self):
            return "%8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f" \
285                 % (self.m1, self.m2, self.l1, self.l2, self.l3, self.l4, \
                    self.theta1c, self.theta2c, self.theta3c)
287
        def printColumnInformation(self):
289         if toggleEnergies or toggleAngles or toggleCoordinates:
                print "%8s" % ("Time"),
291
            if toggleAngles:
293             print "%8s %8s %8s" % ("Theta1", "Theta2", "Theta3"),

295         if toggleCoordinates:
                print "%8s %8s %8s %8s %8s %8s %8s %8s" \
297                 % ("pos2x","pos2y", "pos1x", "pos1y", "pos4x", "pos4y", "pos5x", "pos5y"),

299         if toggleEnergies:
                print "%8s %8s %8s %8s %8s" \
301                 % ("Pot pro", "Pot cou", "Kin pro", "Kin cou", "Tot E"),

303         if toggleImpact:
                print "%8s" % ("Impact"),
305
            if toggleEnergies or toggleAngles or toggleCoordinates:
307             print

309
        # Print summary of the simulation
311     def printSummary(self, s):
            print "%s %9.5f %15.11f %10.6g" \
313                 % (s, self.maxLength / self.maxLengthTeoretic, \
                    self.maxLength, timestepInit)
315
    # The actual code
317
    # Importer de nÃ¸dvendige biblioteker.
319 from optparse import OptionParser
    import math
321 import sys

323 # Try to load the gnuplot functionality, is it not available
    # disable the functionality.
325 try:
        import Gnuplot
327     import Gnuplot.PlotItems
        import Gnuplot.funcutils
329 except:
        pass
331
    # Define option parser, inspired by tutorial on
333 # http://docs.python.org/lib/optparse-tutorial.html
    usage = "usage: %prog [options] arg1 arg2"
335 parser = OptionParser()
    parser.add_option("--gnuplot",
337             action="store_true",
             default=False,
339          dest="gnuplot",
             help="Toggle gnuplot animations")
341 parser.add_option("--energies",
             action="store_true",
343          default=False,
             dest="energies",
345          help="Show info on location of kinetic and potential energy.")
    parser.add_option("--position",
347          type="float", nargs=9, dest="position",
             metavar="m1 m2 l1 l2 l3 l4 t1 t2 t3",
349          help="Set initial position of the trebuchet. "\
             "m1: mass of projectile, "\
351          "m2: mass of counterweight, "\
             "l1: length from pivot to sling attach, "\
353          "l2: length of xx, "\
             "l3: length of xx, "\
355          "l4: length of xx, "\
             "t1: angle of main arm (in radians), "\
357          "t2: angle of xxx (in radians), "\
```

```
            "t3: angle of xxx (in radians)",
359         default=(1.0, 100.0, 4.0, 1.0, 4.0, 1.0, 0.780, 0.000, -1.570))
    parser.add_option("--angles",
361         action="store_true",
            default=False,
363         dest="angles",
            help="Show coordinate values in each iteration.")
365 parser.add_option("--coordinates",
            action="store_true",
367         default=False,
            dest="coordinates",
369         help="Show coordinate values in each iteration.")
    parser.add_option("--impact",
371         action="store_true",
            default=False,
373         dest="impact",
            help="Show info on impact location.")
375 parser.add_option("--redrawtime",
            default=0.001,
377         type="float",
            dest="redrawtime",
379         help="Set time between redraws in the animation.")
    parser.add_option("--steplength",
381         default=0.00001,
            type="float",
383         dest="steplength",
            help="Set steplength in seconds.")
385 parser.add_option("--gravitationalacceleration",
            default=9.82,
387         type="float",
            dest="g",
389         help="Set gravitational acceleration.")
    parser.add_option("--nosummary",
391         action="store_false",
            default=True,
393         dest="summary",
            help="Display summary (initial position and found throwing length).")
395 parser.add_option("--reductionfactor",
            default=1,
397         type="int",
            dest="divider",
399         metavar="factor",
            help="Reduce amount of output by factor.")
401
    # Parse the given options
403 (options, args) = parser.parse_args()

405 # Move options from the commandline to actual variables in the
    # system
407 toggleGnuplot       = options.gnuplot
    toggleImpactPrint   = options.impact
409 toggleEnergies      = options.energies
    toggleImpact        = options.impact
411 toggleAngles        = options.angles
    toggleCoordinates   = options.coordinates
413 timestepInit        = options.steplength
    divider             = options.divider
415 redrawTime          = options.redrawtime
    g                   = options.g
417
    # If animation is enabled, initialize the gnuplot functionality.
419 if toggleGnuplot:
        try:
421         # Initialize an instanze of gnuplot
            gp = Gnuplot.Gnuplot()
423         gp.clear()
            gp('set data style linespoint')
425         gp('set xrange [-9:7]')
            gp('set yrange [-5:11]')
427         gp('set size square')
        except:
429         # Initialization failed, disable animations.
            toggleGnuplot = false
431
    # Initialize some variables
433 lastTimePlottet = 0
    counter = 0
435
    # Make a trebuchet object.
437 testblide = Blide()

439 # Set initial position of the trebuchet, values are fetched
    # from the commandline.
441 (testblide.m1, testblide.m2,
            testblide.l1, testblide.l2, testblide.l3, testblide.l4,
443         testblide.theta1c, testblide.theta2c, testblide.theta4c)\
                = options.position
445
    testblide.printColumnInformation()
447
```

```
    # Save the original state of the trebuchet
449 parm = testblide.parameterString()

451 # Print the actual state of the trebuchet
    testblide.printPosition()
453
    # Take many steps with the method.
455 # After each step print information regarding the state of the
    # trebuchet.
457 while testblide.time < 1:
        testblide.step()
459     testblide.printPosition()

461 # Display a summary of the simulation (if selected)
    if options.summary:
463     testblide.printSummary(parm)
```

Litteratur

[Buckingham(1914)] Buckingham, Edgar. 1914. On physically similar systems; illustrations of the use of dimensional equations. *Physical Review* 4(4):345–357.

[Chevedden(2000)] Chevedden, Paul E. 2000. The invention of the counterweight trebuchet: A study in cultural diffusion. *Dumbarton Oaks Papers* (54). URL `http://www.doaks.org/DOP54/DP54ch4.pdf`.

[Chevedden and Eigenbrod(1995)] Chevedden, Paul E., and Les Eigenbrod. 1995. The trebuchet. *Scientific American* 273(1):66.

> God introduktion til blidens historie, og hvordan maskinen har haft indflydelse på verdenen. Beskriver blidens virkemåde og hvordan forskellige forbedringer har gjort bliden mere effektiv.

[Christiansen(1999)] Christiansen, Edmund. 1999. *Numerisk analyse*. Institut for Matematik og Datalogi, Syddansk Universitet.

> Bog anvendt til faget "Numerisk Analyse". Fra bogen har jeg hentet en fjerde ordens Runge–Kutta metode, samt teknikken til Richardson ekstrapolation.

[Lewis and Papadimitriou(1998)] Lewis, Harry J., and Christos H. Papadimitriou. 1998. *Elements of the theory of computation, second edition*. Prentice Hall.

> Bog anvendt til kurset "Automatteori og beregnelighed"(DM17) i efteråret 2005.

[Price(2003)] Price, James F. 2003. Dimensional analysis of models and data sets. *American Journal of Physics* 71(5):437–447.

[Shinbrot et al.(1992)Shinbrot, Grebogi, Wisdom, and Yorke] Shinbrot, Troy, Celso Grebogi, Jack Wisdom, and James A. Yorke. 1992. Chaos in a double pendulum. *American Journal of Physics* 60(6):491–499.

> Eksperimentel og nummerisk undersøgelse af et dobbelt pendul. Der påvises kaotisk opførsel i begge systemer. Desuden vises det at usikkerheden vokser med en faktor af størrelsesordnen e^8 pr. sekund.

[Siano(2001)] Siano, Donald B. 2001. Trebuchet mechanics. URL `http://www.algobeautytreb.com/trebmath35.pdf`.

> Grundig gennemgang af blidens mekanik, er udgangspunktet for dette bachelor projekt. Siano gennemgår forskellige simuleringsmodeller for en blide, og undersøger hvordan de forskellige forbedringer (slynge og hængslet ballast) påvirker virkningsgraden.

[Vemming(1996)] Vemming, Peter. 1996. Fortidens dræbermaskine slår til igen. *Illustreret Videnskab* (3):34–37.

Den artikel der indledte min interesse for blider. Artiklen beskriver, hvordan bliden blev benyttet, og nogle af blidens fordele (træfsikkerhed m.m.), men fortæller ikke særlig meget om mekanikken bag.

[Wikipedia(2006)] Wikipedia. 2006. Trebuchet. URL `http://en.wikipedia.org/wiki/Trebuchet`.

[Wraa(1997)] Wraa, Morten. 1997. *Hamilton-Jacobi-ligningen i generel fysik*. Master's thesis.

Speciale rapport omhandlende Hamilton–Jacobi ligningen. Indeholder flere gode eksempler og er derfor meget relevant i denne opgave.