



Rotasjon av stive legemer

Av

Håkon Harnes, Vetle Harnes og Jørgen Hollum

Veileder: Hans Jakob Rivertz

Prosjekt i TDAT3024
Matematikk og fysikk valgfag

Norges Teknisk-Naturvitenskapelige Universitet

Trondheim, 30. oktober 2020

Håkon Harnes

Håkon Harnes

Vetle Harnes

Vetle Harnes

Jørgen Hollum

Jørgen Hollum

Innholdsfortegnelse

Innledning	1
Teori	2
2.1 Variabler	2
2.2 Matriser	2
2.2.1 Diagonalmatrisen	2
2.2.2 Identitetsmatrisen	3
2.2.3 Orthogonale matriser	3
2.2.4 Determinant av 3x3 matrise	3
2.2.5 Rotasjonsmatriser	4
2.3 Den spesielle orthogonale gruppen	4
2.4 Rotasjon av stiftt legeme	5
2.5 Trehetsmomentet	5
2.5.1 Trehetsmomentet til et roterende legeme	6
2.6 Dreiemoment	6
2.6.1 Dreiemomentet til et roterende legeme	7
2.7 Symmetriske og asymmetriske legemer	7
2.8 Dzhanibekov effekten	8
2.9 Koordinatsystem	9
2.10 Energien til et roterende legeme	10
2.11 Bevegelseslikningene	10
2.11.1 Bevegelsesligninger for roterende koordinatsystem	10
2.12 X-matrissen	11
2.13 Omega og sigma	12
2.14 Eulers metode	13
2.14.1 Feilanalyse	13
2.15 Runge–Kutta av fjerde orden (RK4)	14
2.15.1 Feilanalyse	15
2.16 Runge–Kutta–Fehlberg metode (RK45)	15
2.16.1 Feilanalyse	18
Metode	19
3.1 Oppgave 1	19
3.1.1 Oppgave a	19
3.1.2 Oppgave b	19
3.1.3 Oppgave c	20
3.2 Oppgave 2	22
3.3 Oppgave 3	24
3.4 Oppgave 4	25

3.4.1	Runge-Kutta (RK4)	25
3.4.2	Runge-Kutta-Fehlberg (RKF45)	26
3.5	Oppgave 5	27
3.6	Oppgave 6	27
Resultat		28
4.1	Oppgave 1	28
4.1.1	Oppgave a	28
4.1.2	Oppgave b	29
4.1.3	Oppgave c	29
4.2	Oppgave 2	30
4.3	Oppgave 3	30
4.4	Oppgave 4	31
4.4.1	Runge-Kutta (RK4)	31
4.4.2	Runge-Kutta-Fehlberg (RKF45)	32
4.5	Oppgave 5	33
4.5.1	Oppgave a	33
4.5.2	Oppgave b	33
4.5.3	Oppgave c	34
4.6	Oppgave 6	35
4.6.1	Oppgave a	36
4.6.2	Oppgave b	37
4.6.3	Oppgave c	38
4.7	Numerisk feil	39
4.7.1	RK4, $h = 0.1$	40
4.7.2	RK4, $h = 0.01$	43
4.7.3	RK4, $h = 0.001$	46
4.7.4	RKF45	49
Diskusjon		53
5.1	Rotasjon av kulelegemet	53
5.1.1	Numerisk feil	53
5.2	Rotasjon av T-nøkkelen	53
5.2.1	Rotasjon om 1. akse	54
5.2.2	Rotasjon om 2. akse	56
5.2.3	Rotasjon om 3. akse	56
5.2.4	Energien	57
5.2.5	Trehetsmomentet	57
5.3	Numerisk feil	57
5.3.1	Energien	57
5.3.2	Feilen i X	58

5.3.3	Sammenhengen mellom energien og feilen i X	58
5.3.4	Den akkumulative feilen i X	58
5.3.5	Sammenhengen mellom den akkumulative feilen i X og energien	58
5.3.6	Steglengden i RK4	59
5.3.7	Steglengden i RKF45	59
Konklusjon		60
Referanser		62
Vedlegg		63
6.1	Kildekode	63
6.2	Prosjektplan	101
Erklæring		103
7.1	Håkon Harnes	103
7.2	Vetle Harnes	104
7.3	Jørgen Hollum	105

Figurer

1	T-nøkkelen er et asymmetrisk legeme. På grunn av håndtaket vil tregheitsmomentet være størst over 3. aksen (z-aksen), minst for 2. aksen (y-aksen) og mellomliggende for 1. aksen. [1, kapittel 7 figur (2)]	8
2	Komponentene til X , rotasjon om 1. akse	36
3	Komponentene til X , rotasjon om 2. akse	37
4	Komponentene til X , rotasjon om 3. akse	38
5	Energien plottet mot tiden når $h = 0.1$	40
6	Feilen i X plottet mot tiden når $h = 0.1$	40
7	Feilen i X plottet mot energien når $h = 0.1$	41
8	Akkumulativ feil i X plottet mot energien når $h = 0.1$	41
9	X_1 plottet mot feilen i X når $h = 0.1$	42
10	X_1 plottet mot akkumulativ feil i X når $h = 0.1$	42
11	Energien plottet mot tiden når $h = 0.01$	43
12	Feilen i X plottet mot tiden når $h = 0.01$	43
13	Feilen i X plottet mot energien når $h = 0.01$	44
14	Akkumulativ feil i X plottet mot energien når $h = 0.01$	44
15	X_1 plottet mot feilen i X når $h = 0.01$	45
16	X_1 plottet mot akkumulativ feil i X når $h = 0.01$	45
17	Energien plottet mot tiden når $h = 0.001$	46
18	Feilen i X plottet mot tiden når $h = 0.001$	46
19	Feilen i X plottet mot energien når $h = 0.001$	47
20	Akkumulativ feil i X plottet mot energien når $h = 0.001$	47
21	X_1 plottet mot feilen i X når $h = 0.001$	48
22	X_1 plottet mot akkumulativ feil i X når $h = 0.001$	48
23	Energien plottet mot tiden for RKF45	49
24	Feilen i X plottet mot tiden for RKF45	49
25	Feilen i X plottet mot energien for RKF45	50
26	Akkumulativ feil i X plottet mot energien for RKF45	50
27	X_1 plottet mot feilen i X for RKF45	51
28	X_1 plottet mot akkumulativ feil i X for RKF45	51
29	Steglengden plottet mot tiden for RKF45	52
30	Steglengden plottet X_1 for RKF45	52

Tabeller

1 Fysiske egenskaper for T-nøkkelen 20

Innledning

Denne rapporten er en omfattende analyse av prosjektoppgaven vi ble gitt i mattedelen av faget TDAT3024. Prosjektoppgaven tar for seg to forskjellige legemer som roterer rundt sitt eget massesenter, uten noen form for ytre påvirkning.

I rapporten tar vi for oss legemets orientering i rommet. Helt konkret skal vi se nærmere på rotasjonen, energien og bevegelsen til både et symmetrisk (kule) og et asymmetrisk stivt legeme (T-nøkkel). Vi skal også ta i bruk flere numeriske metoder for å finne en tilnærmet løsning, da vi ikke kan løse det sistnevnte problemet eksakt. Videre sammenligner vi presisjonen og feilen til de ulike numeriske metodene. Avslutningsvis grafer vi den numeriske løsningen, og gir en tolkning av denne. Dette er supplementert med en animasjon av T-nøkkelen som skal gi leseren en visuell og intuitiv forståelse av bevegelsen til T-nøkkelen.

De numeriske metodene er implementert i matlab, mens oppgaver som omhandler eksakt løsning er løst for hånd.

Teori

2.1 Variabler

Tabellen under gir en oversikt over de viktigste variablene for problemstillingen:

Symbol	Forklaring
$X(t)$	Funksjonen av endringen til koordinatsystemet per tidsenhet
$W(t)$	Approksimasjon til X
I	Treghetsmoment
Id_{3x3}	Identitetsmatrisen (3x3)
\vec{L}	Dreiemoment
$\vec{\omega}$	Rotasjonsvektor
Ω	Rotasjonsmatrise
h	Steglengde
ρ	Massetetthet

2.2 Matriser

Denne seksjonen tar for seg generell matriseregning som ble benyttet i prosjektet.

2.2.1 Diagonalmatrisen

En diagonal matrise er en kvadratisk matrise hvor alle elementene i matrisen er 0, bortsett fra diagonalen.

$$X = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix}$$

2.2.2 Identitetsmatrisen

En identitetsmatrise er en diagonal matrise hvor alle komponentene på diagonalen er 1. Når man multipliserer en matrise med identitetsmatrisen, blir svaret utgangsmatrisen.

$$Id_{3 \times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.2.3 Ortogonale matriser

En matrise er ortogonal hvis produktet av matrisen med dens transposerte matrise er lik identitetsmatrisen.

$$XX^T = Id_{3 \times 3} \quad (2.3.1)$$

2.2.4 Determinant av 3x3 matrise

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{33} \\ a_{31} & a_{22} & a_{33} \end{bmatrix} = a_{11} \cdot \det \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} - a_{12} \cdot \det \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} + a_{13} \cdot \det \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

2.2.5 Rotasjonsmatriser

Rotasjonsmatrisen over 1., 2. og 3. akse er gitt ved:

$$R_1(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) \\ 0 & \sin(t) & \cos(t) \end{bmatrix} [2]$$

$$R_2(t) = \begin{bmatrix} \cos(t) & 0 & \sin(t) \\ 0 & 1 & 0 \\ -\sin(t) & 0 & \cos(t) \end{bmatrix} [2]$$

$$R_3(t) = \begin{bmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} [2]$$

2.3 Den spesielle ortogonale gruppen

Den spesielle ortogonale gruppen, også kalt Lie-gruppen, kan beskrives som en gruppe 3x3 matriser hvor disse er ortogonale med den egenskapen at determinanten er lik 1. Vi får da

$$XX^T = Id_{3x3}$$

hvor

$$\det X = 1$$

[1, kapittel 3.4 avsnitt (17)]

2.4 Rotasjon av stivt legeme

Et viktig valg i fysikken, er valg av koordinatsystem, og dets samhandling med legement man regner på. Når man arbeider med roterende legemer, vil det ofte lønne seg å bruke et koordinatsystem med massesenter i origo. Hvis legemet ikke er i ro, men beveger seg i en konstant retning og fart, vil posisjonen til massesenteret være gitt ved:

$$\vec{r}_{C.M}(t) = \frac{1}{M} \int \int \int_M \vec{r}(t) dm$$

[1, kapittel 3 avsnitt (1)]

2.5 Treghetsmomentet

Treghetsmoment er gjenstandens motstand mot å få sin rotasjonshastighet endret, og er med på å bestemme hvor stor spinn og bevegelsesenergi gjenstanden har i rotasjonen. Motstanden avhenger av massedistribusjonen til legemet.

$$I = mr^2$$

Ut i fra formelen ser vi at det jo lengre bort fra rotasjonsaksen massen er, jo høyre motstand blir det for legemet å rotere legemet rundt rotasjonsaksen.

2.5.1 Treghetsmomentet til et roterende legeme

Treghetsmomentets partiellderiverte av høyeste orden er gitt ved:

$$I_{xx} = \int \int \int_M (y_b^2 + z_b^2) dm$$

$$I_{yy} = \int \int \int_M (x_b^2 + z_b^2) dm$$

$$I_{zz} = \int \int \int_M (x_b^2 + y_b^2) dm$$

$$I_{xy} = I_{yx} = \int \int \int_M (x_b y_b) dm$$

$$I_{xz} = I_{zx} = \int \int \int_M (x_b z_b) dm$$

$$I_{yz} = I_{zy} = \int \int \int_M (y_b z_b) dm$$

[1, kapittel 3.2 avsnitt (13)]

2.6 Dreiemoment

Dreiemoment er et mål på hvor stor kraft som skal til for å få et legeme til å rotere om en akse. I likhet med at kraft er det som skal til for at et objekt skal akselerere i en bevegelse, er dreiemoment kraften som skal til for at et objekt skal oppnå vinkel akselrasjon. Dette er gitt ved:

$$\tau = F \times r$$

Ved rotasjonsbevegelse blir dreiemomentet et produkt av treghetsmomentet og vinkelakselrasjonen:

$$\tau = I\alpha$$

2.6.1 Dreiemomentet til et roterende legeme

Dreiemomentet til legemet om massesenteret er gitt ved.

$$\vec{L}_c = \int \int \int_M \vec{r}_c(t) \times (\vec{\omega}(t) \times \vec{r}_c(t)) dm$$

[1], kapittel 3.1 avsnitt (5).

Denne er uavhengig av hvilket som helst valgt koordinatsystem med origo i massesenteret.

Den partiellederiverte for dreiemomentet er gitt ved:

$$L_x = I_{xx}\omega_x - I_{xy}\omega_y - I_{xz}\omega_z$$

$$L_y = -I_{yx}\omega_x - I_{yy}\omega_y - I_{yz}\omega_z$$

$$L_z = -I_{zx}\omega_x - I_{zy}\omega_y - I_{zz}\omega_z$$

[1, kapittel 3.2 avsnitt (12)]

Dermed blir dreiemomentet:

$$\vec{L} = XI\vec{\omega}_b$$

[1, kapittel 3.3 avsnitt (17)]

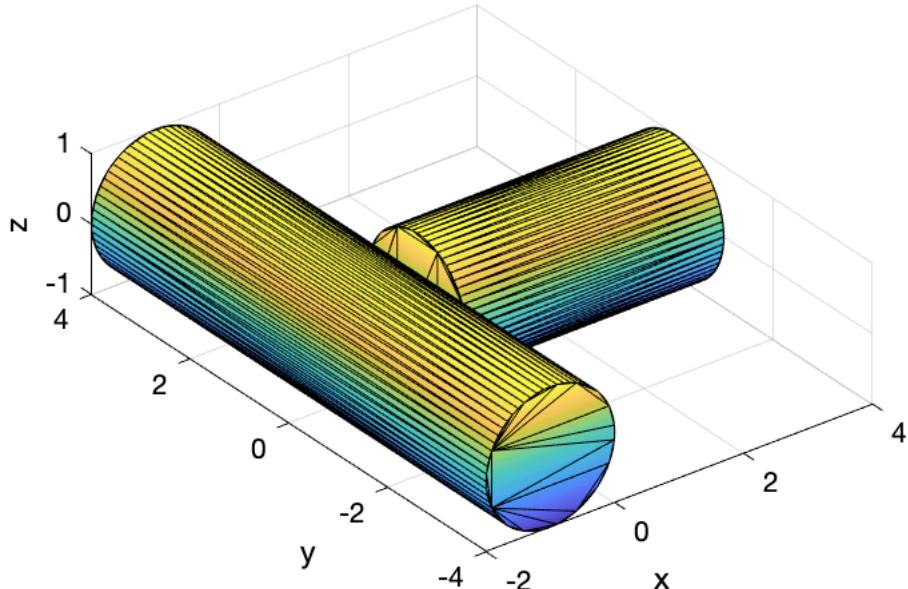
2.7 Symmetriske og asymmetriske legemer

Veritasium definerer et legeme som har tre ulike størrelser treghetsmoment fordelt over tre akser som en *asymmetric top*. Gjennom denne seksjonen referer vi til dette som et asymmetrisk legeme. [3]

Et asymmetrisk legemet er et legeme hvor det finnes tre ulike rotasjonsakser med ulike treghetsmoment. For T-nøkkelen vil 1. aksen ha den mellomliggende verdien for treghetsmomentet. 2. aksen vil ha lavest treghetsmoment. 3. aksen vil ha høyest treghetsmoment. Dette er en følge av at massen er ulikt distribuert i T-nøkkelen. Dersom vi roterer rundt 1. aksen vil et velkjent fenomen oppstå, nemlig Dzhanibekov effekten.

Symmetriske legemer vil ha samme treghetsmoment for alle de tre ulike rotasjonsaksene. En kule er et eksempel på dette. Da vil det ikke oppstå noen Dzhanibekov effekt.

2.8 Dzhanibekov effekten



Figur 1: T-nøkkelen er et asymmetrisk legeme. På grunn av håndtaket vil treghetsmomentet være størst over 3. aksen (z-aksen), minst for 2. aksen (y-aksen) og mellomliggende for 1. aksen. [1, kapittel 7 figur (2)]

Dersom vi roterer rundt 1. aksen har vi kun sentripetalkrefter som akselererer de største massene mot senteret. Dette gjør at legemet holder en uniform sirkulær bevegelse. Dersom vi velger at koordinatsystemet roterer med legemet vil vi kunne fokusere på centrifugalkraften. Sentrifugalkraft dyster massene ut mot periferien av legemet, hvor da kreftene er propor-

sjonale med deres avstand fra y-aksen. Disse kraftene er utlignet hos de store massenes sentripetalkrefter som virker innover i legemet.

Når legemet forskyves i 2. akse retning, vil de små massene oppleve en centrifugalkraften som er proporsjonal til deres distanse fra 1. aksen. Spenningskrekter forsikrer om at de små massene virker ortogonalt på de store massene, og siden de store massene fortsatt spinner omrent i samme posisjon som før tvinges de små massene til å ligge i xy-planet.

Sentrifugalkraften forårsaker en akselerasjon på de små massene som blir større og større jo lengre massen flyttes unna 1. aksen, og til slutt ender disse opp med å bytte side. Akselrasjonen i denne bevegelsen øker og har sin største verdi i midten av bevegelsen, altså når de små massene er lengst unna 1. aksen. Videre vil akselrasjonen avta da de beveger seg nærmere 1. aksen. Denne type bevegelses mønster repeteres på ubestemt tid, med legemet som vender seg fram og tilbake med jevne mellomrom.

2.9 Koordinatsystem

Et koordinatsystem hvor aksene er hovedaksene til treghetsmomentet til legemet lønner seg fordi dreimomentet er uavhengig av koordinatsystemet vi bruker:

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} \frac{dL_x}{dt} \\ \frac{dL_y}{dt} \\ \frac{dL_z}{dt} \end{bmatrix} + \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \times \begin{bmatrix} L_x \\ L_y \\ L_z \end{bmatrix}$$

[1, kapittel 3.1 avsnitt (10)]

Vi antar at legemet ikke påvirkes av noen ytre krefter og får da $\begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix} = 0$.

2.10 Energien til et roterende legeme

Siden vi ser på et system uten noe ytre påvirkning, vil energien i systemet være bevart gjennom hele prosessen. Den kinetiske rotasjonsenergien til legemet er videre gitt ved dreiemomentmoment \vec{L} og rotasjonsvektor $\vec{\omega}$. Dette er gitt ved formelen:

$$K = \frac{1}{2} \vec{L} \cdot \vec{\omega} \quad (2.9.1)$$

[1, kapittel 3.3 avsnitt (14)]

2.11 Bevegelseslikningene

Konstant akselrasjon	konstant vinkelaskelrasjon
$s = vt$	$\theta = \omega t$
$v = v_0 + at$	$\omega = \omega_0 + \alpha t$
$s = v_0 t + \frac{1}{2}at^2$	$\theta = \omega_0 t + \frac{1}{2}\alpha^2 t$
$s = \frac{v+v_0}{2}t$	$\theta = \frac{\omega+\omega_0}{2}t$
$v^2 - v_0^2 = 2as$	$\omega^2 - \omega_0^2 = 2\alpha\theta$

2.11.1 Bevegelsesligninger for roterende koordinatsystem

Siden koordinatsystemet roterer sammen med legemet, er enhetsvektorene som vi definerer som $\hat{i}_b(t)$, $\hat{j}_b(t)$ og $\hat{k}_b(t)$ som betyr at koordinatsystemet er en funksjon av tiden.

$$\begin{aligned} \frac{d\hat{i}_b}{dt} &= \omega_z \hat{j}_b - \omega_y \hat{k}_b \\ \frac{d\hat{j}_b}{dt} &= \omega_x \hat{k}_b - \omega_y \hat{i}_b \\ \frac{d\hat{k}_b}{dt} &= \omega_y \hat{i}_b - \omega_x \hat{j}_b \end{aligned}$$

[1, kapittel 3.4 avsnitt (15)]

2.12 X-matrisen

Når man skriver ut alle likningene i kapittel 2.11.1, ender man opp med 9 koblede differensiallikninger. Disse er lettere å implementere som en 3x3 matrise X, der hver søyle består av komponenten til en akse i koordinatsystemet. Vi lar derfor $X = [x_{ij}]$ bestå av søylevektorene til \hat{i}_b , \hat{j}_b og \hat{k}_b , der:

$$\hat{i}_b = \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix}, \hat{j}_b = \begin{bmatrix} x_{12} \\ x_{22} \\ x_{32} \end{bmatrix}, \hat{k}_b = \begin{bmatrix} x_{13} \\ x_{23} \\ x_{33} \end{bmatrix}$$

[1, kapittel 3.4 avsnitt (16)]

som betyr at:

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

X matrisen har også de spesielle egenskapene at $X^T X = Id_{3x3}$, $\det X = 1$ og tilhører Lie-gruppen.

2.13 Omega og sigma

Rotasjonsvektoren $\vec{\omega}_b$ er gitt på formen:

$$\omega_b = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

[1, kapittel 3.4 avsnitt (17)]

Den er gitt ved:

$$\vec{\omega}_b = (XI)^{-1}\vec{L}$$

[1, kapittel 3.4 avsnitt (19)]

Videre trenger vi i visse tilfeller en konstant Ω , som kan utledes fra ω_b , ved hjelp av å putte de tilhørende verdiene inn i matrisen:

$$\Omega = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

[1, kapittel 3.4 avsnitt (18)]

Samme fremgangsmåte gjelder også når man skal gå fra $\vec{\sigma}$ til Σ , som brukes når man skal løse differensial likningen med RK45.

2.14 Eulers metode

En differensiallikning er gitt på formen

$$y' = \frac{dy}{dt} = f(t, y)$$

Eulers metode er en numerisk metode for approksimering av løsningen til en ordinær differensiallikning. Metoden tar som utgangspunkt i en initialverdi (t_0, y_0) og steglengde h :

$$w_0 = y(t_0)$$

$$w_{i+1} = w_i + h f(t_i, w_i)$$

[1, kapittel 4.1 avsnitt (23)]

Hvor $t_i = t_0 + hi$ og $w_i \approx y(t_i)$.

2.14.1 Feilanalyse

For hver iterasjon av t finner vi gradienten i punktet. Ser vi på hvert verdi par vil grafen og punktene som dannes av Eulers metode vil vi komme til å se at de begge vil følge hverandre, men at Eulers metode vil ha et feilestimat og derfor ligge nogenlunde utenfor grafen.

Global avkortningsfeil er den akkumelerte feilen fra første antall steg i. Denne er gitt ved $g_i = |w_i - y_i|$, gitt at man har en eksakt løsning på problemet. Global avkortnings feilen på Eulers metode vil alltid være proposjonal med h , som gir oss en feil på $O(h)$

Lokal avkortningsfeil er feilen som oppstår ved et enslig steg, tatt i betraktning forrige løsnings tilnærming w_i som startpunkt. $e_{i+1} = |w_{i+1} - z(t_{i+1})|$ gitt at vi har en eksakt løsning. Feilen på lokal avkortning vil alltid være proporsjonal med h^2 , som gir oss en feil på $O(h^2)$.

2.15 Runge–Kutta av fjerde orden (RK4)

RK4 er en metode for å tilnærmet løse ordinære differensial likninger. Dette er en trinnvis metode, som betyr at den kun trenger en startverdi for å utføres. Som en 4. ordens metode gir den også en mye høyere presisjon enn metoder som Eulersmetode eller trapesmetoden. Denne metoden regner ut den neste verdien av:

$$w_{n+1}$$

ved å bruke

$$w_n$$

og et vektet gjennomsnitt av fire forskjellige inkremente over intervallet h med større vektlegging på de midterste stigningstallene, der:

s_1 er et inkrement som er basert på stigningstallet i begynnelsen av intervallet

s_2 og s_3 er et inkrement basert på stigningstallet i midten av intervallet

s_4 er et inkrement basert på stigningstaller i slutten av intervallet

Runge-Kutta av fjerde orden metoden er gitt ved:

$$w_{i+1} = w_i + h/6(s_1 + 2s_2 + 2s_3 + s_4)$$

der

$$\begin{aligned}s_1 &= f(t_i, w_i) \\s_2 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_1\right) \\s_3 &= f\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_2\right) \\s_4 &= f(t_i + h, w_i + hs_3)\end{aligned}$$

2.15.1 Feilanalyse

Siden RK4 er en 4. ordens funksjon, vil feilen være gitt ved $O(h^4)$, noe som er markant bedre enn Eulers metode som vi brukte tidligere i oppgaven. Å finne feilen i Runge-Kutta kan være utfordrende, men metoder som å regne ut en mer nøyaktig løsning ved å bruke $\frac{h}{2}$ og deretter sammenligne med den opprinnelige løsningen for h kan gi deg et godt estimat. En annen metode er å sammenligne svaret med en 5. ordens Runge-Kutta for å få et estimat av feilen, men gjengangen er at man må gjøre en ny og mer krevende utregning for å få et svar på feilen ved denne metoden.

2.16 Runge–Kutta–Fehlberg metode (RK45)

Problemet med Runge-Kutta metoden, er at det er vanskelig å avgjøre nøyaktigheten til løsningen. Vanligvis regner man på problemet to ganger med steglengder h og $h/2$ eller en høyere orden, men hvis dette ikke er godt nok må man gjenta dette som er tid og ressurskrevende. Det er dette Runge-Kutta-Fehlberg (RK45) metoden prøver å løse.

RK45 har en prosedyre for å avgjøre om riktig steglengde h er tatt i bruk mellom hvert steg. Dette gjøres ved at man ved hvert steg gjør to forskjellige Runge-Kutta approksimasjoner, en 4. ordens og en 5. ordens, og sammenligner disse. Hvis begge disse løsningene er like opp til en gitt nøyaktighet, blir dette steglengden som brukes. Hvis de ikke stemmer opp til en gitt nøyaktighet derimot, reduseres steglengden og man prøver på nytt. Hvis svaret blir mer nøyaktig enn ønsket, økes steglenden [4]. Grunnen til at dette er gunstig, er fordi 4. og 5. ordens løsningene bruker nesten bare de samme variablene, noe som betyr at å regne ut 5. ordens metoden for å sammenlikne ikke krever så mye mer datakraft. (bare utregning av s_6) Dette gjør RK45 til en rask og nøyaktig numerisk metode ved små endringer, som også kan takle store endringer.

RK45 bruker et såkalt Butcher tabell, og er definert ved:

c	A
	B

Innsatt med verdiene til A , B og c får vi:

c_i	A_{ij}				
0					
$\frac{1}{4}$	$\frac{1}{4}$				
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$		
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56450}$	$-\frac{9}{50}$
					$\frac{2}{55}$

[1, kapittel 4.5 avsnitt (39)]

Da er sigma er gitt ved:

$$\begin{aligned}
 s_1 &= hf(t_i + c_1 h, w_i) \\
 s_2 &= hf(t_i + c_2 h, w_i + a_{21}s_1) \\
 s_3 &= hf(t_i + c_3 h, w_i + a_{31} + a_{32}s_2) \\
 s_4 &= hf(t_i + c_4 h, w_i + a_{41}s_1 + a_{42}s_2 + a_{43}s_3) \\
 s_5 &= hf(t_i + c_5 h, w_i + a_{51}s_1 + a_{52}s_2 + a_{53}s_3 + a_{54}s_4) \\
 s_6 &= hf(t_i + c_6 h, w_i + a_{61}s_1 + a_{62}s_2 + a_{63}s_3 + a_{64}s_4 + a_{65}s_5)
 \end{aligned}$$

[4, avsnitt (28)]

Så finner vi en tilnærming av initialverdi problemet ved å bruke en 4. ordens Runge-Kutta der:

$$w_{k+1} = w_k + b_{11}s_1 + b_{12}s_2 + b_{13}s_3 + b_{14}s_4 + b_{15}s_5$$

[4, avsnitt (29)]

Deretter finner man en bedre verdi for løsningen ved å bruke en 5. ordens Runge-Kutta der:

$$z_{k+1} = z_k + b_{21}s_1 + b_{22}s_2 + b_{23}s_3 + b_{24}s_4 + b_{25}s_5 + b_{26}s_6$$

[4, avsnitt (30)]

Videre kan man finne den optimale steglengden h ved å gange skalaret k_s ganget med den nåværende steglengden, som er gitt ved:

$$k_s = \left(\frac{tol \times h}{2|z_{k+1} - w_{k+1}|} \right)^{\frac{1}{4}}$$

[4, avsnitt (31)]

Der $tol \times h$ er den gitte feil-kontroll toleransen.

2.16.1 Feilanalyse

Siden vi bruker en 5. ordens Runge-Kutta på en RK4 til å definere feilen i hvert steg, vil vi få en feilforstørring $O(h^5)$.

RK45 lar oss videre sette en toleranse for feilen i utregningen ved hjelp av å bruke en 5. ordens Runge-Kutta til å sammenligne forskjellene på hvert steg man tar. Dette lar deg oppnå et veldig presist estimat på feilen, der man kan definere feiltoleransen selv. Dette gjør RK45 til en mer sofistikert metode enn RK4 og Eulers metode.

Metode

Denne seksjonen tar for seg hvordan teorien ble anvendt i praksis for å løse oppgavene i prosjektet.

3.1 Oppgave 1

Oppgaven gikk ut på å implementere følgende funksjoner:

- (a) Eksponentialfunksjonen for et roterende legeme
- (b) Energien til et roterende legeme
- (c) Trehetsmomentet til T-nøkkelen

3.1.1 Oppgave a

Eksponentialfunksjonen for et roterende legeme er gitt ved:

$$\exp(h\Omega) = Id_{3x3} + (1 + \cos(\omega h)) \frac{\Omega^2}{\omega^2} + \sin(\omega h) \frac{\Omega}{\omega} \quad (3.1.1)$$

Metoden `Exp.e(h, o)` implementerer ligning (3.1.1). For å teste metoden kan vi bruke ligning (2.3.1). Den er implementert i metoden `Exp.test(X, TOL)`.

3.1.2 Oppgave b

Energien til det roterende legeme er gitt ved ligning (2.9.1). Metoden `Energy.calculate(L, w)` implementerer (2.9.1) nøyaktig slik den står.

3.1.3 Oppgave c

Treghetsmomentet til T-nøkkelen er gitt ved:

$$I_{xx} = \frac{M_1 R_1^2}{4} + \frac{M_1 L_1^2}{12} + \frac{M_2 R_2^2}{2} \quad (3.1.1)$$

$$I_{yy} = M_1 R_1^2 + \frac{M_2 L_2^2}{4} + \frac{M_1 R_1^2}{2} + \frac{M_2 R_2^2}{4} + \frac{M_2 L_2^2}{12} \quad (3.1.2)$$

$$I_{zz} = M_1 R_1^2 + \frac{M_2 L_2^2}{4} + \frac{M_1 R_1^2}{4} + \frac{M_1 L_1^2}{12} + \frac{M_2 R_2^2}{4} + \frac{M_2 L_2^2}{12} \quad (3.1.3)$$

Matrisen er diagonal. Altså er $I_{xy} = I_{xz} = I_{yz} = 0$. Treghetsmomentet blir da:

$$A = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

T-nøkkelen har vi følgende egenskaper:

Egenskap	Symbol	Verdi
Radius, håndtak	R_1	1.0 cm
Radius, sylinder	R_2	1.0 cm
Lengde, håndtak	L_1	8.0 cm
Lengde, sylinder	L_2	4.0 cm
Massetetthet, T-nøkkel	ρ	6.7 gram/cm ³
Masse, T-nøkkel	M	$12\pi\rho$ gram

Tabell 1: Fysiske egenskaper for T-nøkkelen

For å regne ut trehetsmomentet til T-nøkkelen trengte vi M_1 og M_2 . Disse fant vi ved å regne ut volumet til hver sylinder, for så å multiplisere med massetetheten.

$$M_1 = V_1 \rho$$

$$M_2 = V_2 \rho$$

Volumene til sylinderne er gitt ved:

$$V_1 = \pi R_1^2 L_1$$

$$V_2 = \pi R_2^2 L_2$$

Dermed er massene M_1 og M_2 gitt ved:

$$M_1 = \pi R_1^2 L_1 \rho \tag{3.1.4}$$

$$M_2 = \pi R_2^2 L_2 \rho \tag{3.1.5}$$

Klassen `THandle` har variabler `r1`, `r2`, `l1`, `l2`, `m`, `p`, som er de fysiske egenskapene til t-nøkkelen fra tabell 1. Metoden `THandle.calculateMomentOfInertia()` regner først ut M_1 og M_2 v.h.a. henholdsvis ligning (3.1.4) og (3.1.5). Deretter bruker metoden ligningene (3.1.1), (3.1.2) og (3.1.3) for å regne ut trehetsmomentet til t-nøkkelen.

3.2 Oppgave 2

Oppgaven gikk ut på å løse likningssettet:

$$\vec{\omega}_b = (XI)^{-1}\vec{L}$$

$$\frac{dX}{dt} = F(X) = X\Omega$$

Hvor legemet er en kule. $X(0)$, I og \vec{L} var gitt ved:

$$X(0) = I = Id_{3x3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \vec{L} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Siden legemet er en kule, vet vi at den bare vil rotere over én akse, som vi kan se utfra dreiemomentet \vec{L} er 1. aksen. Siden X er en del av Lie-gruppen, er matrisen ortogonal. En rotasjon rundt 1. aksen vil som nevnt i kapittel (2.3.5) være på formen:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} [2]$$

Regner ut determinanten til X :

$$\begin{aligned} \det(X) &= \det \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} = 1 \cdot \det \begin{bmatrix} c & -s \\ s & c \end{bmatrix} - 0 \cdot \det \begin{bmatrix} 0 & -s \\ 0 & -c \end{bmatrix} + 0 \cdot \det \begin{bmatrix} 0 & s \\ 0 & c \end{bmatrix} \\ &= 1 \cdot \det \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = 1(c \cdot c - s \cdot (-s)) = c^2 + s^2 \end{aligned}$$

Siden X er inneholdt i Lie-gruppen vet vi at $\det(X) = 1$:

$$\det(X) = 1 \Rightarrow c^2 + s^2 = 1$$

Utifra formelen over, oppbygningen til rotasjonsmatriser over 1. aksen[2] og den trigonometriske identiteten på formen $\sin^2(t) + \cos^2(t) = 1$, kommer vi frem til at:

$$c^2 + s^2 \Rightarrow \sin^2(t) + \cos^2(t) = 1$$

Dermed vil s og c være gitt ved:

$$s = \sin(t)$$

$$c = \cos(t)$$

Innsatt i X -matrisen gir dette:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) \\ 0 & \sin(t) & \cos(t) \end{bmatrix}$$

Deretter kan vi finne ut $\vec{\omega}_b$ ved å sette inn løsningen for X i formelen vår.

Siden I er gitt ved identitetsmatrisen, vi $(XI)^{-1} = X^{-1}$, og siden X er ortogonal får vi:

$$X^{-1} = X^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & \sin(t) \\ 0 & -\sin(t) & \cos(t) \end{bmatrix}$$

Får da videre ved matrisemultiplikasjon at:

$$\vec{\omega}_b = (XI)^{-1}\vec{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & \sin(t) \\ 0 & -\sin(t) & \cos(t) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

3.3 Oppgave 3

Oppgaven gikk ut på å implementere en variant av Eulers metode. En mulighet er:

$$W_0 = X_0$$

$$W_{i+1} = W_i + hW_i\Omega_i$$

Problemet her er at W_i ikke er ortogonal. Derfor ble følgende variant implementert:

$$W_0 = X_0 \tag{3.3.1}$$

$$W_{i+1} = W_i \exp(h\Omega_i) \tag{3.3.2}$$

Ligningene (3.3.1) og (3.3.2) er implementert i metoden `EulersMethod.solve(X0, I, L)`. Metoden er en del av klassen `EulersMethod`, som inneholder steglengden og antall iterasjoner.

3.4 Oppgave 4

Oppgaven gikk ut på å implementere en variant av Runge-Kutta, og eventuelt Runge-Kutta-Fehlberg.

3.4.1 Runge-Kutta (RK4)

Metoden er som følger:

$$\begin{aligned}\sigma_1 &= I^{-1} W_i^T \vec{L} \\ \sigma_2 &= I^{-1} \exp(-(h/2)\Sigma_1) W_i^T \vec{L} \\ \sigma_3 &= I^{-1} \exp(-(h/2)\Sigma_2) W_i^T \vec{L} \\ \sigma_4 &= I^{-1} \exp(-h\Sigma_3) W_i^T \vec{L}\end{aligned}$$

Deretter kan approksimasjonen regnes ut:

$$W_{i+1} = W_i \exp((h/6)(\Sigma_1 + 2\Sigma_2 + \Sigma_3 + \Sigma_4)) \quad (3.4.1)$$

For å regne ut et feilestimat trenger vi en bedre approksimasjon for den eksakte løsningen. Her har vi valgt å bruke samme metode som i RKF45, ved å sammenligne en approksimasjon av 4. orden, W_i , med en 5. ordens approksimasjon, Z_i . Utregningen av Z_i er forklart i kapittelet Runge-Kutta-Fehlberg (RKF45).

Ligningen (3.4.1) med tilhørende verdier $\Sigma_1, \dots, \Sigma_4$ er implementert i metoden `RK4.solve(x0, I, L)`. Metoden er en del av klassen `RK4`, som inneholder steglengden og antall iterasjoner.

3.4.2 Runge-Kutta-Fehlberg (RKF45)

Runge-Kutta-Fehlberg benytter seg av Butcher-tabellen fra teoridelen.

Metoden er som følger:

$$\begin{aligned}\sigma_1 &= I^{-1} W_i^T \vec{L} \\ \sigma_2 &= I^{-1} \exp(-ha_{21}\Sigma_1) W_i^T \vec{L} \\ \sigma_3 &= I^{-1} \exp(-h(a_{31}\Sigma_1 + a_{32}\Sigma_2)) W_i^T \vec{L} \\ \sigma_4 &= I^{-1} \exp(-h(a_{41}\Sigma_1 + a_{42}\Sigma_2 + a_{43}\Sigma_3)) W_i^T \vec{L} \\ \sigma_5 &= I^{-1} \exp(-h(a_{51}\Sigma_1 + a_{52}\Sigma_2 + a_{53}\Sigma_3 + a_{54}\Sigma_4)) W_i^T \vec{L} \\ \sigma_6 &= I^{-1} \exp(-h(a_{61}\Sigma_1 + a_{62}\Sigma_2 + a_{63}\Sigma_3 + a_{64}\Sigma_4 + a_{65}\Sigma_5)) W_i^T \vec{L}\end{aligned}$$

Deretter kan 4- og 5. ordens approksimasjon regnes ut:

$$W_{i+1} = W_i \exp(-h(b_{11}\Sigma_1 + b_{12}\Sigma_2 + b_{13}\Sigma_3 + b_{14}\Sigma_4 + b_{15}\Sigma_5 + b_{16}\Sigma_6)) \quad (3.4.2)$$

$$Z_{i+1} = W_i \exp(-h(b_{21}\Sigma_1 + b_{22}\Sigma_2 + b_{23}\Sigma_3 + b_{24}\Sigma_4 + b_{25}\Sigma_5 + b_{26}\Sigma_6)) \quad (3.4.3)$$

Feilen regnes ut slik:

$$\Delta W = W_{i+1} - Z_{i+1} \quad E = \sqrt{\text{trace}(\Delta W^T \Delta W)}$$

Ligningene (3.4.2) og (3.4.3) med tilhørende verdier $\Sigma_1, \dots, \Sigma_6$ er implementert i metoden `RKF45.solve(X0, I, L)`. Metoden er en del av klassen `RKF45`, som inneholder steglengden, antall iterasjoner og toleransen.

3.5 Oppgave 5

Oppgaven gikk ut på å benytte RK4 eller RKF45 for å løse systemet. Eksperimentet ble kjørt med tre ulike verdier for \vec{L} slik at:

- (a) $\omega(0) = [1, 0.05, 0]^T$
- (b) $\omega(0) = [0, 1, 0.05]^T$
- (c) $\omega(0) = [0.05, 0, 1]^T$

Her må \vec{L} regnes ut for hver $\omega(0)$. \vec{L} er gitt ved:

$$\vec{L} = X(0) \cdot I \cdot \omega(0)$$

Der

$$X(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Trehetsmomentet til T-nøkkelen ble regnet ut i oppgave 1c. Dermed kan systemet løses v.h.a. RK4 eller RKF45, som er implementert i matlab.

3.6 Oppgave 6

Oppgaven gikk ut på å tegne opp komponentene til løsningene X fra forrige oppgave som ni funksjoner av tiden. Eventuelt også lage en animasjon av T-nøkkelen som roterer. Komponentene skal grafes mot tiden, bruker derfor at:

$$t(i + i) = t(i) + h \quad t(1) = 0$$

Da vil t og W være samme lengde. $t(i)$ angir tiden i $X(i)$. Da kan t og X plottes mot hverandre. For å animere t-nøkkelen som roterer, kan vi regne ut ytterpunktene til t-nøkkelen for hver W .

Resultat

4.1 Oppgave 1

4.1.1 Oppgave a

Testet metoden med følgende parametere:

$$\Omega = \begin{bmatrix} 0 & -3 & 2 \\ 3 & 0 & -1 \\ -2 & 1 & 0 \end{bmatrix} \quad h = 0.1$$

Da ga metoden `Exp.e(o, h)` følgende matrise:

$$X = \begin{bmatrix} 0.9358 & -0.2832 & 0.2102 \\ 0.3029 & 0.9506 & -0.0680 \\ -0.1805 & 0.1273 & 0.9753 \end{bmatrix}$$

Testet at X at oppfylte ligning (2.3.1) med metoden `Exp.test(X, TOL)`. Med toleranse lik 10^{-12} fikk vi output 1, altså sant. Matrisen X oppfylte altså ligning (2.3.1) innen den gitte toleransen.

4.1.2 Oppgave b

Testet metoden med følgende parametere:

$$\vec{w} = \vec{L} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Da ga metoden `Energy.calculate(L, w)` følgende energi:

$$K = \frac{1}{2}$$

Energien til dette systemet er altså $\frac{1}{2}$.

4.1.3 Oppgave c

Benyttet fysiske egenskaper til t-nøkkelen gitt i tabell 1 for å opprette et objekt av klassen `THandle`. Regnet så ut treghetsmomementet med `tHandle.calculateMomentOfInertia()`

$$I = 10^3 \times \begin{bmatrix} 0.9823 & 0 & 0 \\ 0 & 0.7227 & 0 \\ 0 & 0 & 1.5787 \end{bmatrix} \quad (4.1.1)$$

4.2 Oppgave 2

$X(0)$, I og L var gitt ved:

$$X(0) = I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \vec{L} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Løsningen ble dermed:

$$X(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) \\ 0 & \sin(t) & \cos(t) \end{bmatrix} \quad \vec{\omega} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (4.2.1)$$

4.3 Oppgave 3

Verdiene for $X(0)$, I og L er lik som i oppgave 2. Steglengden og antall iterasjoner er:

$$h = 0.1 \quad n = 10000$$

Da gir `eulersMethod.solve(X0, I, L)` følgende approksimasjon ved $t = 1000$:

$$W = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0 & 0.5624 & -0.8269 \\ 0 & 0.8269 & 0.5624 \end{bmatrix}$$

Den absolutte feilen ved $t = 1000$ er:

$$|X(1000) - W| = 10^{-9} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.1309 & 0.0897 \\ 0 & 0.0897 & 0.1309 \end{bmatrix}$$

Den absolutte feilen i dette tilfellet er i størrelsesordenen 10^{-9} . Energien endret seg ikke.

4.4 Oppgave 4

4.4.1 Runge-Kutta (RK4)

Verdiene for $X(0)$, I og L er lik som i oppgave 2. Steglengden og antall iterasjoner er:

$$h = 0.1 \quad n = 10000$$

Da gir `rk4.solve(X0, I, L)` følgende approksimasjon ved $t = 1000$:

$$W = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0 & 0.5624 & -0.8269 \\ 0 & 0.8269 & 0.5624 \end{bmatrix}$$

Den absolutte feilen ved $t = 1000$ er:

$$|X(1000) - W| = 10^{-12} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.3871 & 0.4192 \\ 0 & 0.4192 & 0.3871 \end{bmatrix}$$

Den absolutte feilen i dette tilfellet er i størrelsesordenen 10^{-12} . Energien endret seg ikke.

4.4.2 Runge-Kutta-Fehlberg (RKF45)

Verdiene for $X(0)$, I og L er lik som i oppgave 2. Steglengden, antall iterasjoner og toleransen er gitt ved:

$$h = 0.1 \quad n = 10000 \quad TOL = 10^{-50}$$

Da gir `rkf45.solve(X0, I, L)` følgende approksimasjon ved $t = 1000$:

$$W = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0 & 0.5624 & -0.8269 \\ 0 & 0.8269 & 0.5624 \end{bmatrix}$$

Den absolutte feilen ved $t = 1000$ er:

$$|X(1000) - W| = 10^{-13} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.1887 & 0.1288 \\ 0 & 0.1288 & 0.1887 \end{bmatrix}$$

Den absolutte feilen i dette tilfellet er i størrelsesordenen 10^{-13} . Energien endret seg ikke.

4.5 Oppgave 5

Steglengden, antall iterasjoner og toleransen ble satt til:

$$h = 0.01 \quad n = 10000 \quad TOL = 10^{-30}$$

4.5.1 Oppgave a

4.5.1.1 RK4

Beregning	Verdi
Endring i energi	8.29×10^{-5} cm
Største feil	5.57×10^{-25} cm

4.5.1.2 RKF45

Beregning	Verdi
Endring i energi	1.14×10^{-5} cm
Største feil	9.89×10^{-31} cm

4.5.2 Oppgave b

4.5.2.1 RK4

Beregning	Verdi
Endring i energi	1.69×10^{-6} cm
Største feil	3.71×10^{-28} cm

4.5.2.2 RKF45

Beregning	Verdi
Endring i energi	8.69×10^{-7} cm
Største feil	8.39×10^{-31} cm

4.5.3 Oppgave c**4.5.3.1 RK4**

Beregning	Verdi
Endring i energi	5.76×10^{-6} cm
Største feil	3.01×10^{-25} cm

4.5.3.2 RKF45

Beregning	Verdi
Endring i energi	3.42×10^{-7} cm
Største feil	5.82×10^{-31} cm

4.6 Oppgave 6

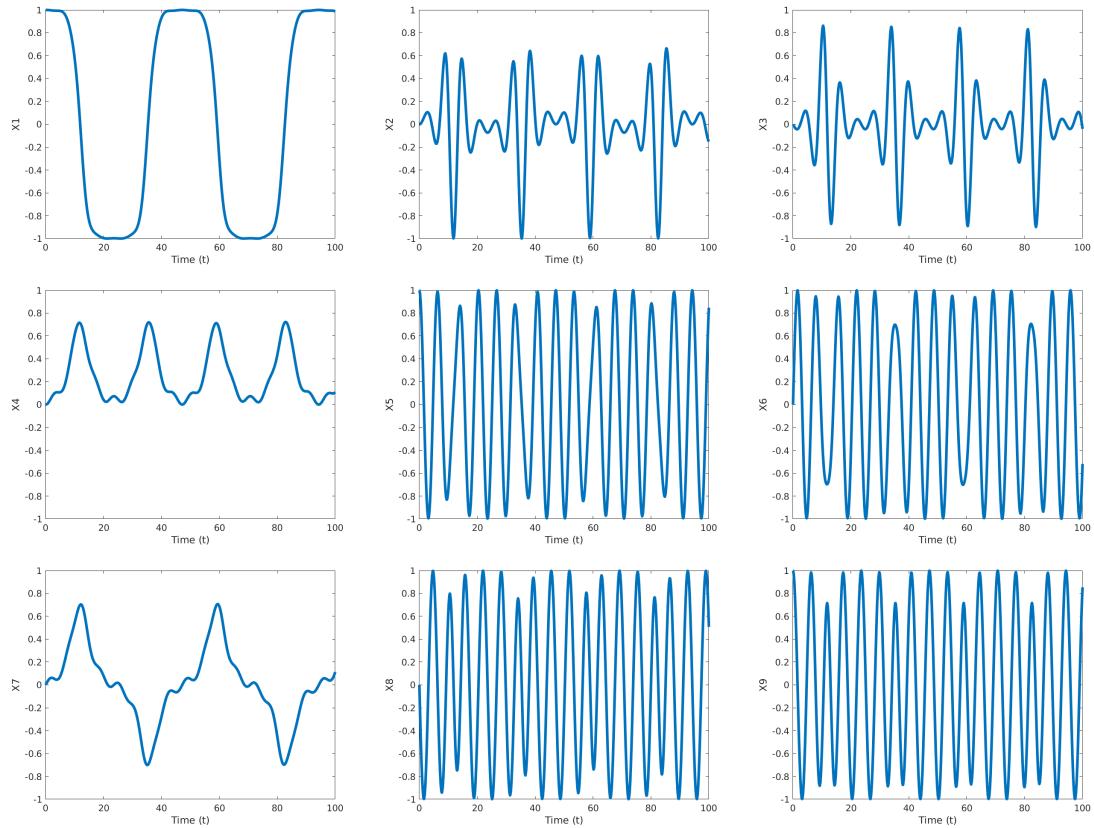
De følgende figurene er sammensatt av ni bilder. Det er et bilde per x-komponent. Bildet er strukturert på samme måte som matrisen X er bygd opp, slik:

$$X = \begin{bmatrix} X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 \\ X_7 & X_8 & X_9 \end{bmatrix}$$

Grafene er produsert v.h.a. RKF45 og presenteres fortløpende på de påfølgende sidene.

4.6.1 Oppgave a

Dette er rotasjon om 1. aks. Da er $\omega(0) = [1, 0.05, 0]^T$.



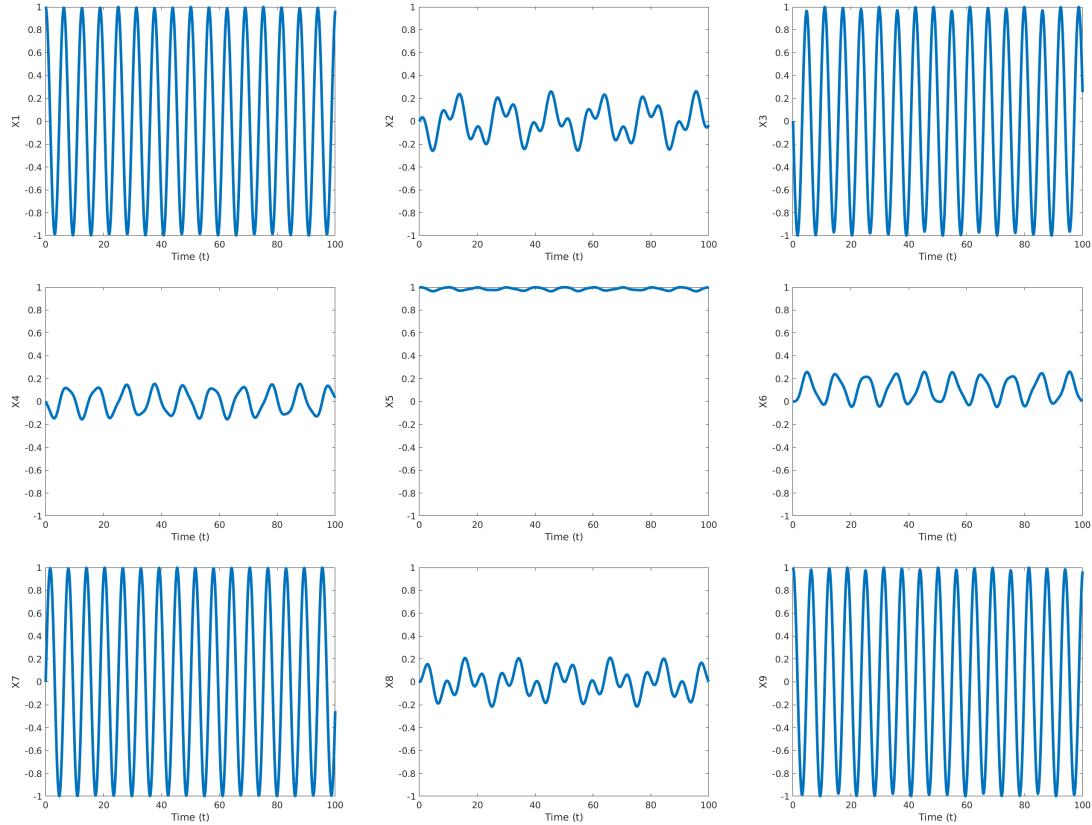
Figur 2: Komponentene til X, rotasjon om 1. aks

Animasjon for dette tilfellet er gitt ved følgende link:

<https://youtu.be/3-jxDQ1voc>

4.6.2 Oppgave b

Dette er rotasjon om 2. akse. Da er $\omega(0) = [0, 1, 0.05]^T$.



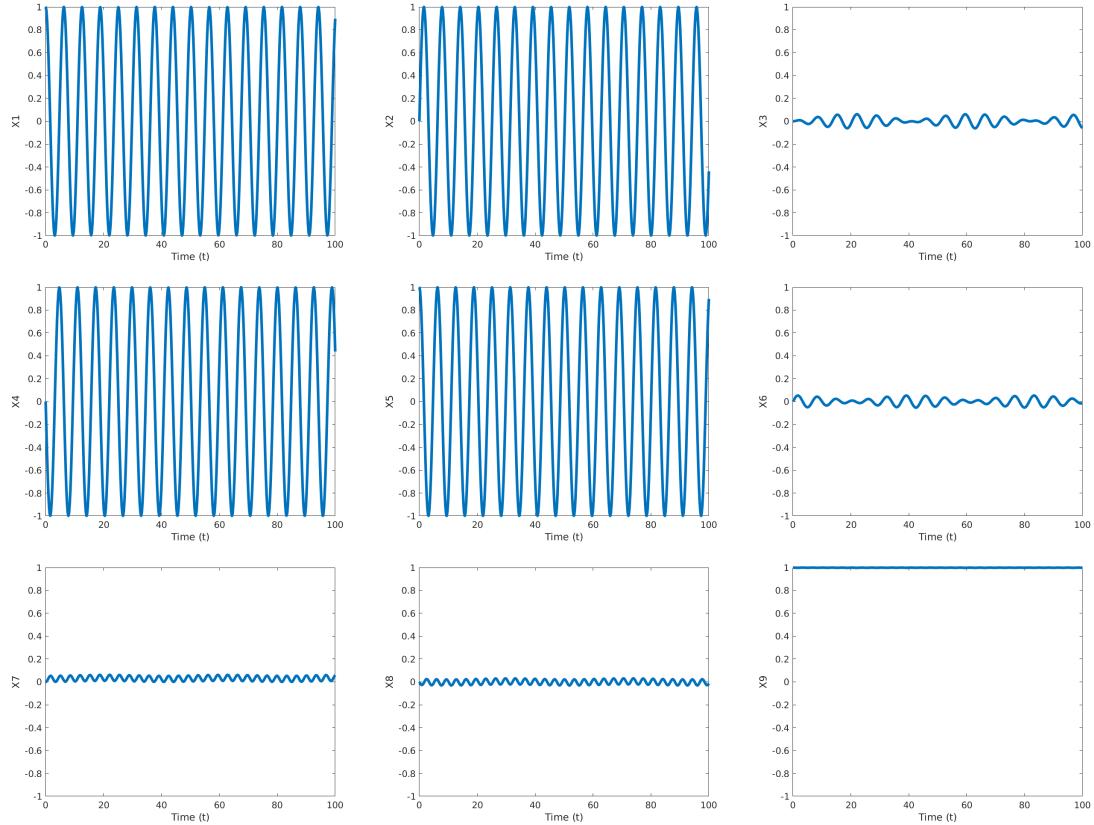
Figur 3: Komponentene til X, rotasjon om 2. akse

Animasjon for dette tilfellet er gitt ved følgende link:

<https://youtu.be/NKI0KE0G0c0>

4.6.3 Oppgave c

Dette er rotasjon om 3. akse. Da er $\omega(0) = [0.05, 0, 1]^T$.



Figur 4: Komponentene til X, rotasjon om 3. akse

Animasjon for dette tilfellet er gitt ved følgende link:

<https://youtu.be/q-iEC1FWBEd>

4.7 Numerisk feil

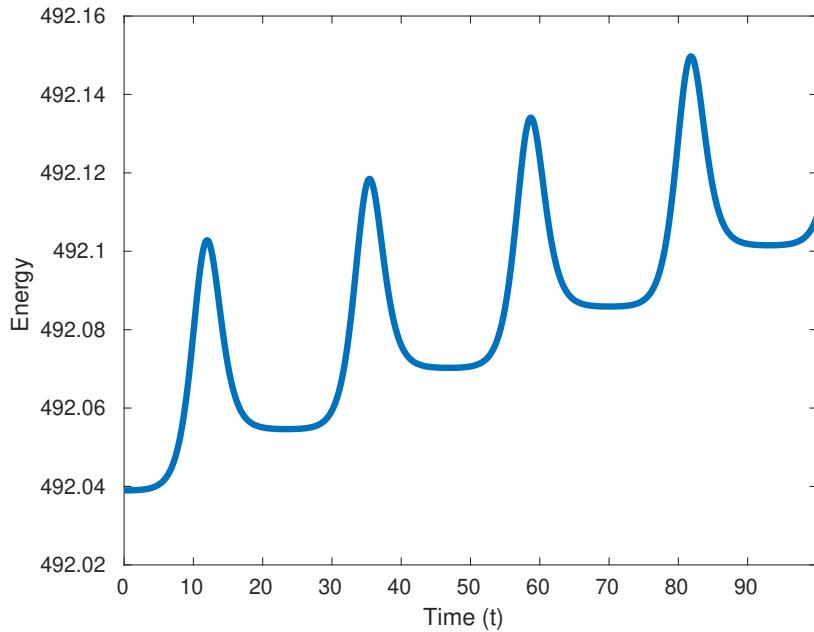
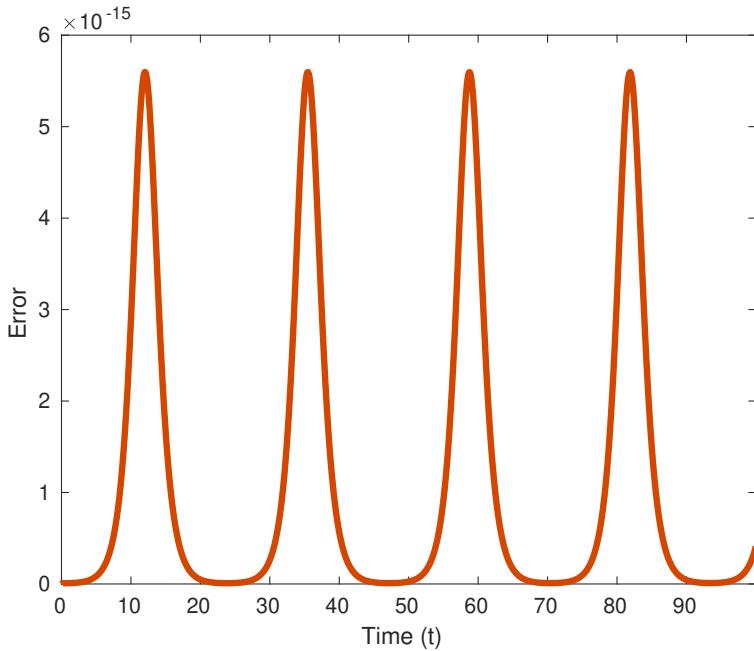
Denne seksjonen tar for seg numerisk feil ved utregningen av approksimasjonen. Det presenteres resultater for både RK4 og RKF45. RKF45 tilpasser steglengden for å oppnå en gitt toleranse. RK4 har blitt kjørt med tre ulike verdier for steglengden:

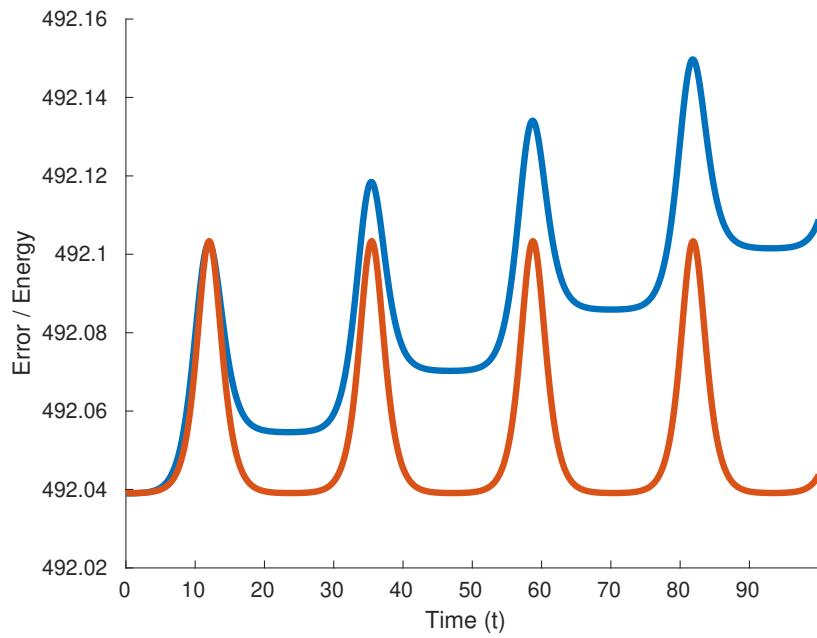
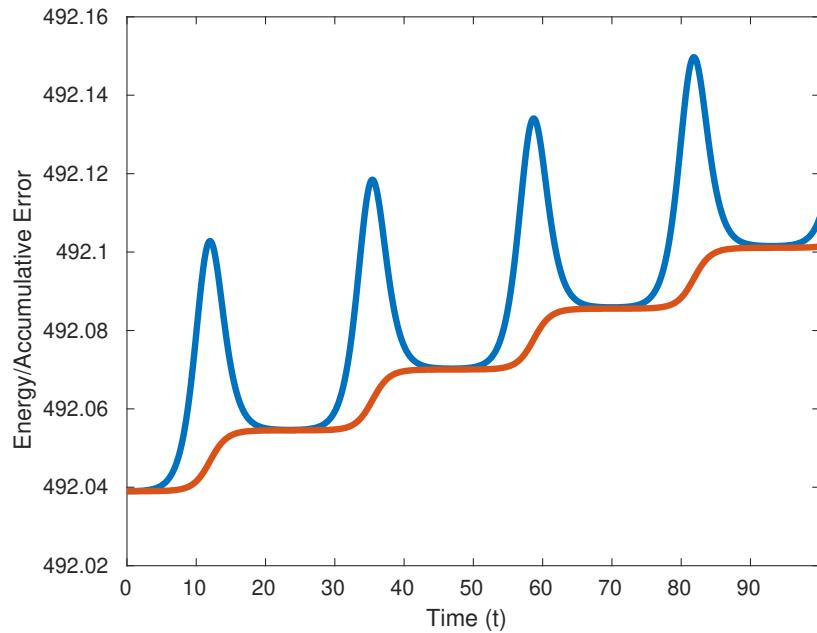
$$h = 0.1 \quad h = 0.01 \quad h = 0.001$$

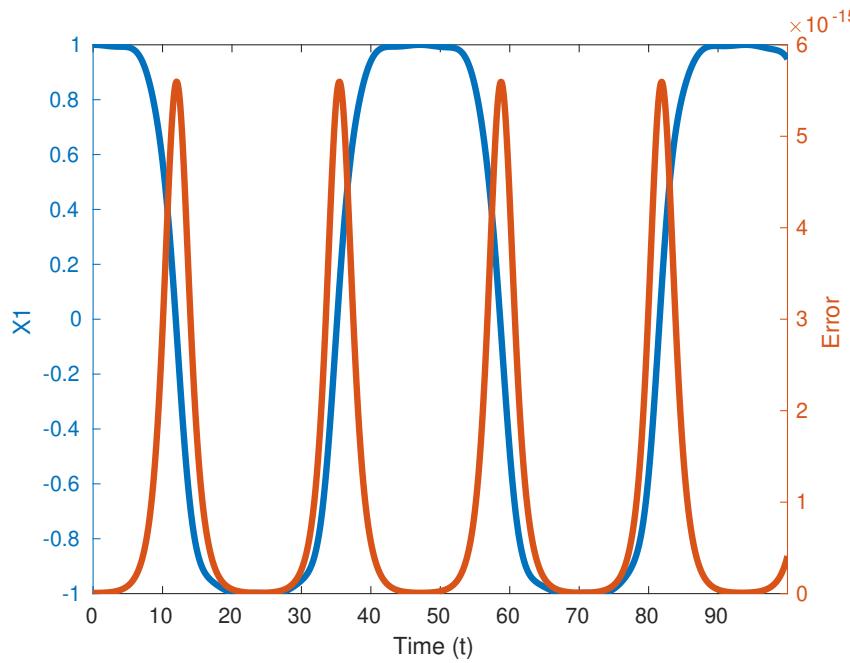
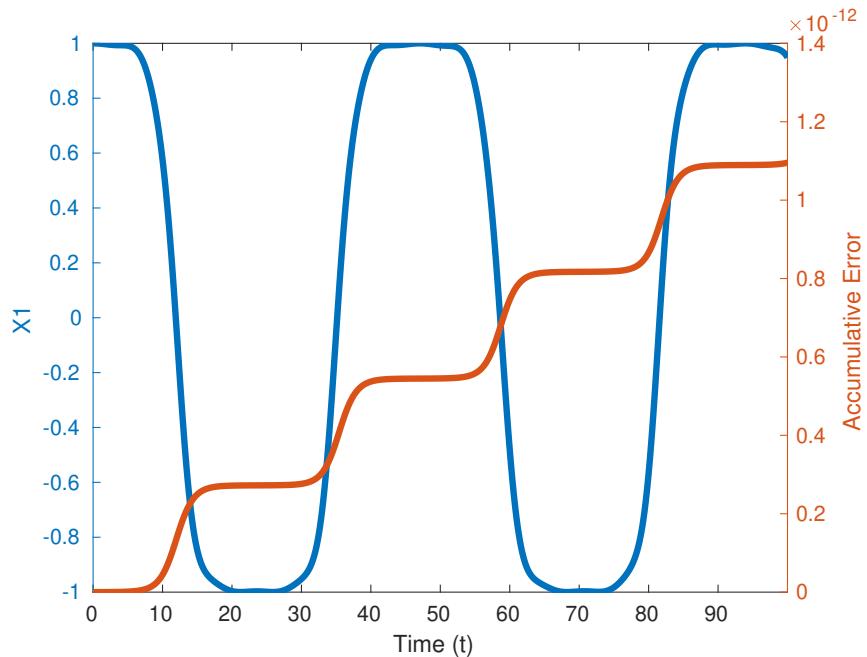
Grafene er produsert for rotasjon om 1. akse, altså der rotasjonsvektoren er:

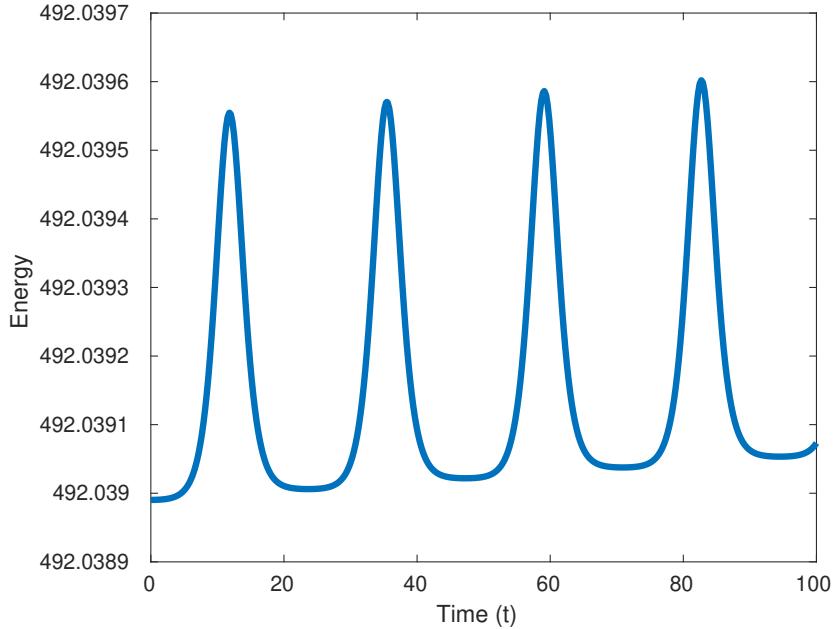
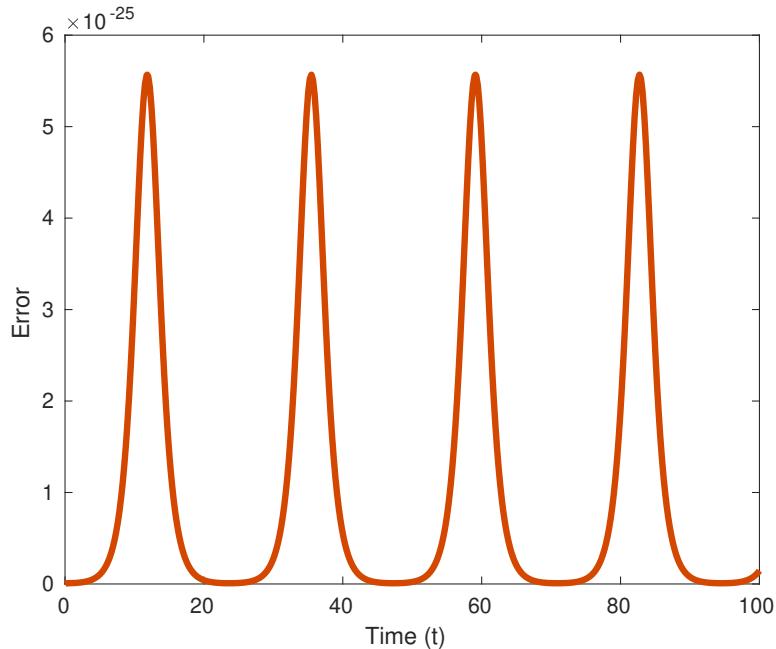
$$\omega(0) = [1, 0.05, 0]^T$$

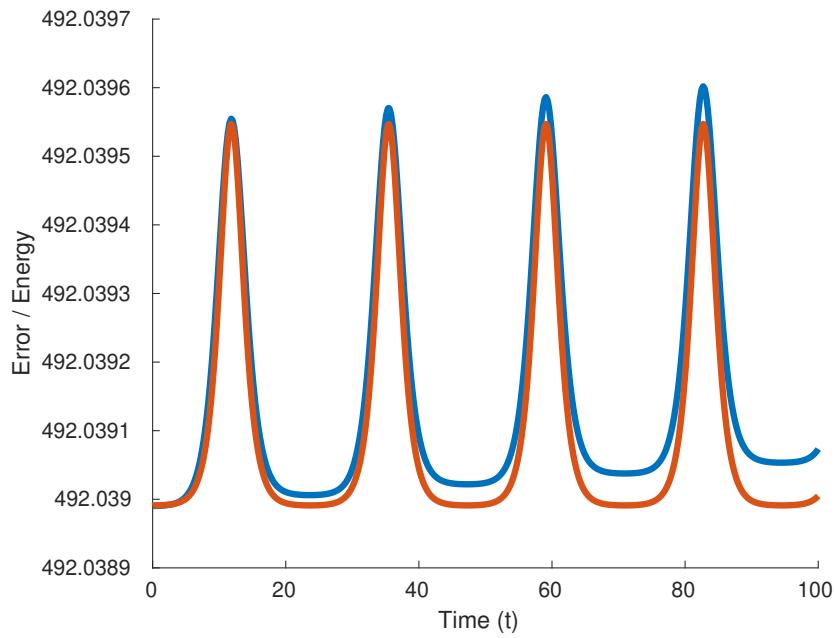
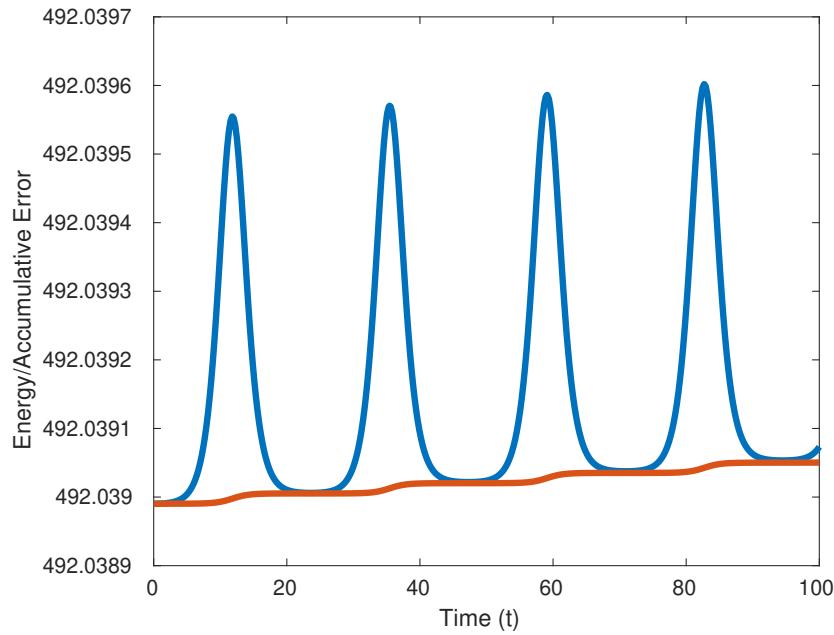
Resultatene presenteres fortløpende på de påfølgende sidene.

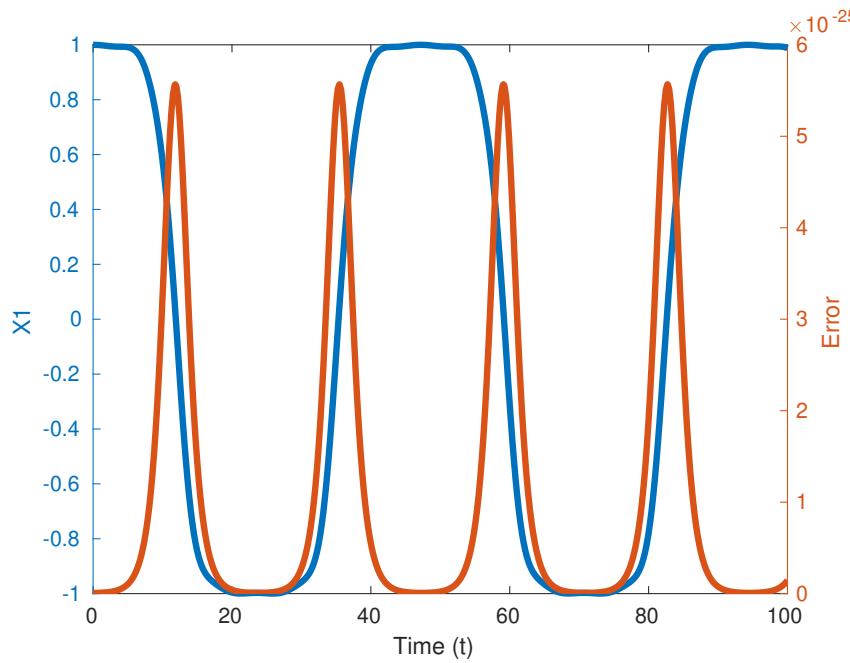
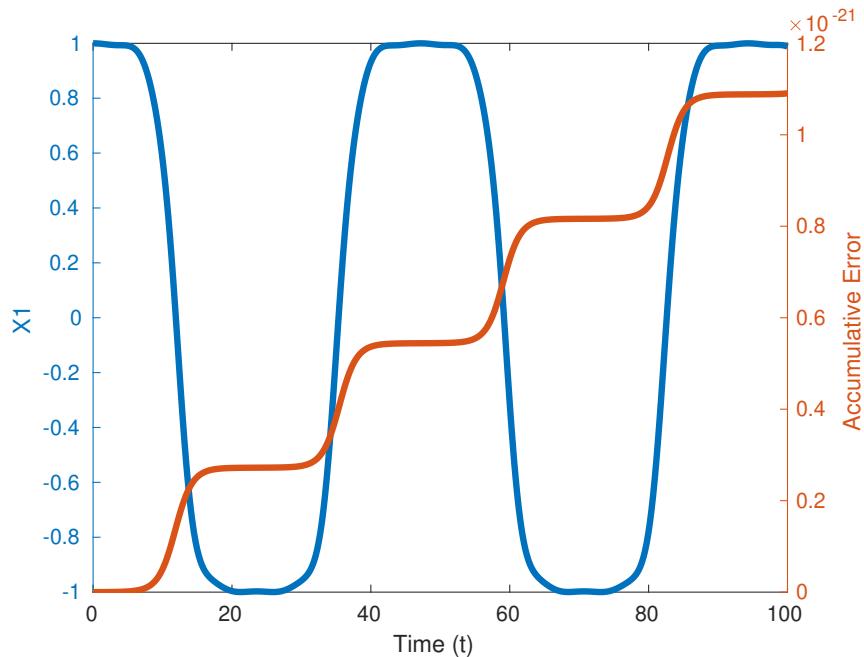
4.7.1 RK4, $h = 0.1$ Figur 5: Energien plottet mot tiden når $h = 0.1$ Figur 6: Feilen i X plottet mot tiden når $h = 0.1$

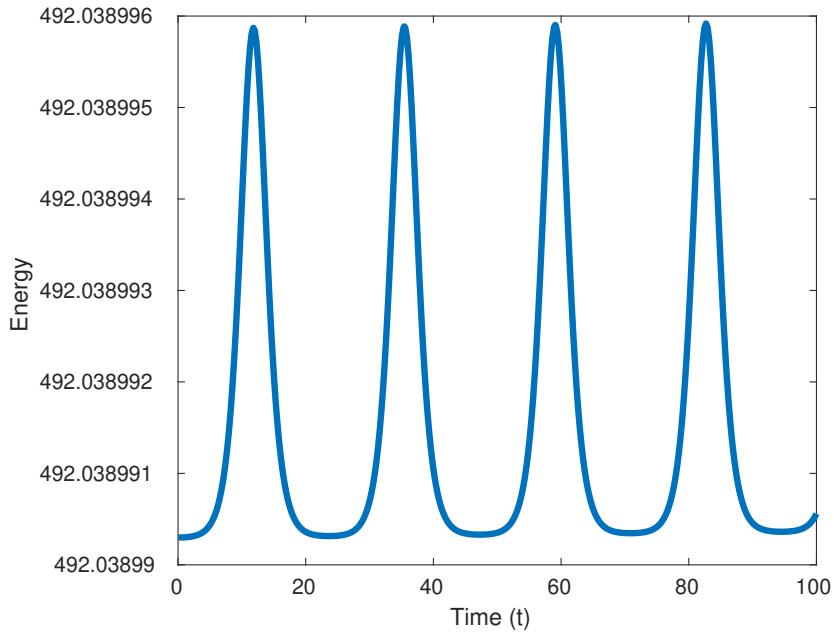
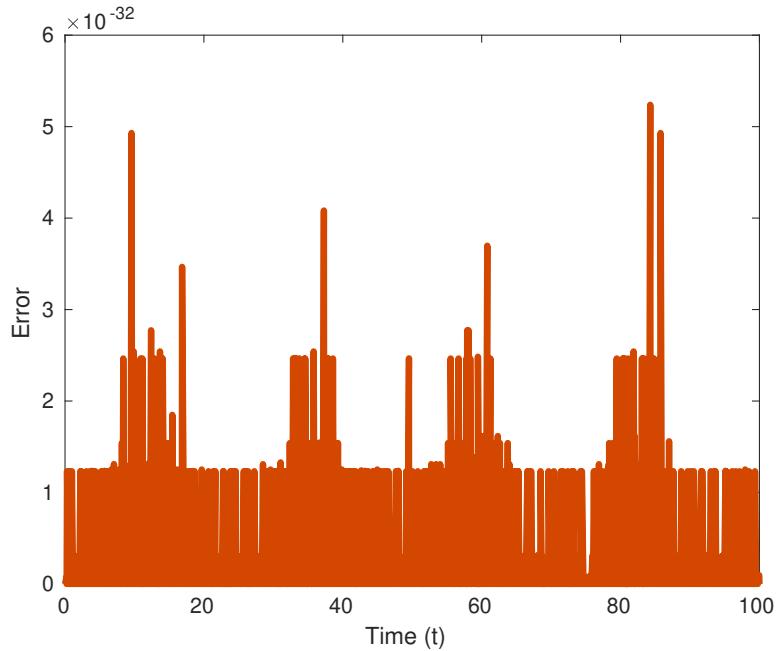
Figur 7: Feilen i X plottet mot energien når $h = 0.1$ Figur 8: Akkumulativ feil i X plottet mot energien når $h = 0.1$

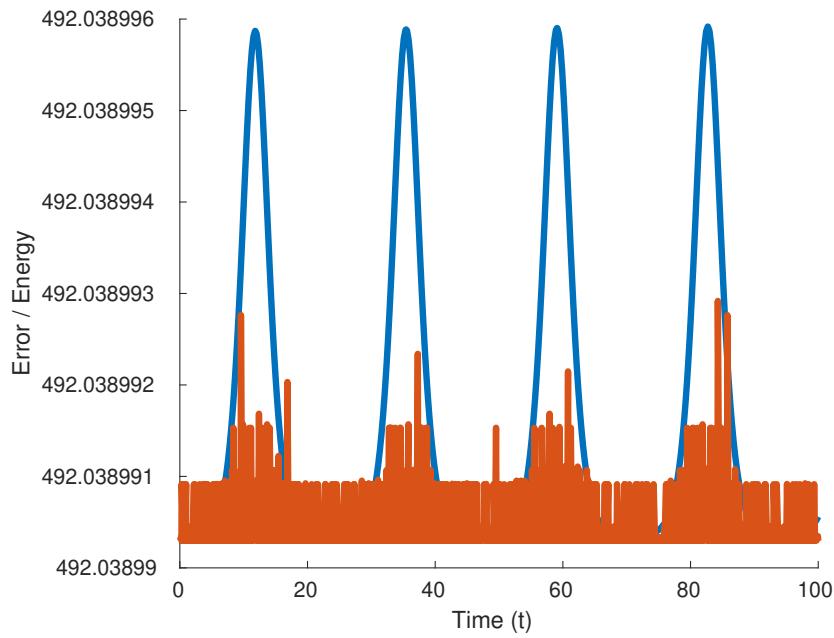
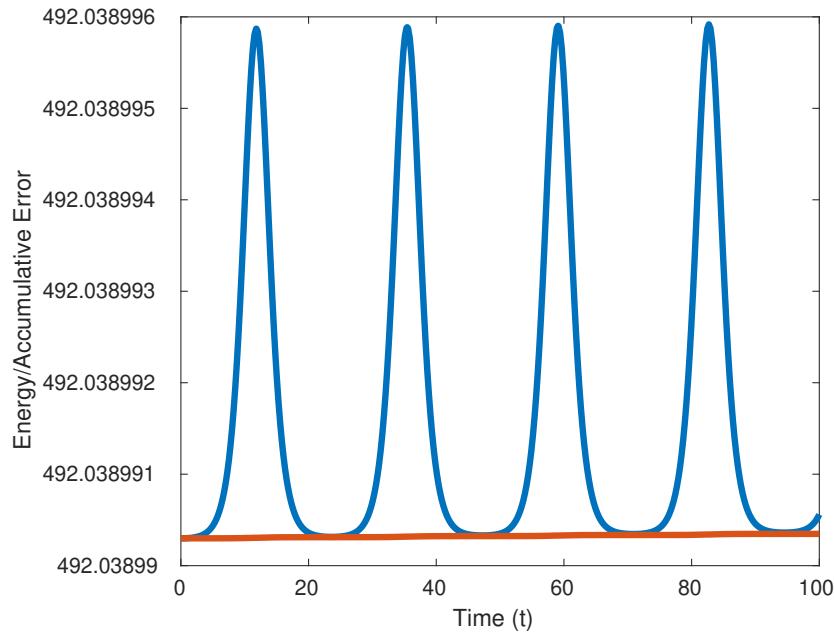
Figur 9: X_1 plottet mot feilen i X når $h = 0.1$ Figur 10: X_1 plottet mot akkumulativ feil i X når $h = 0.1$

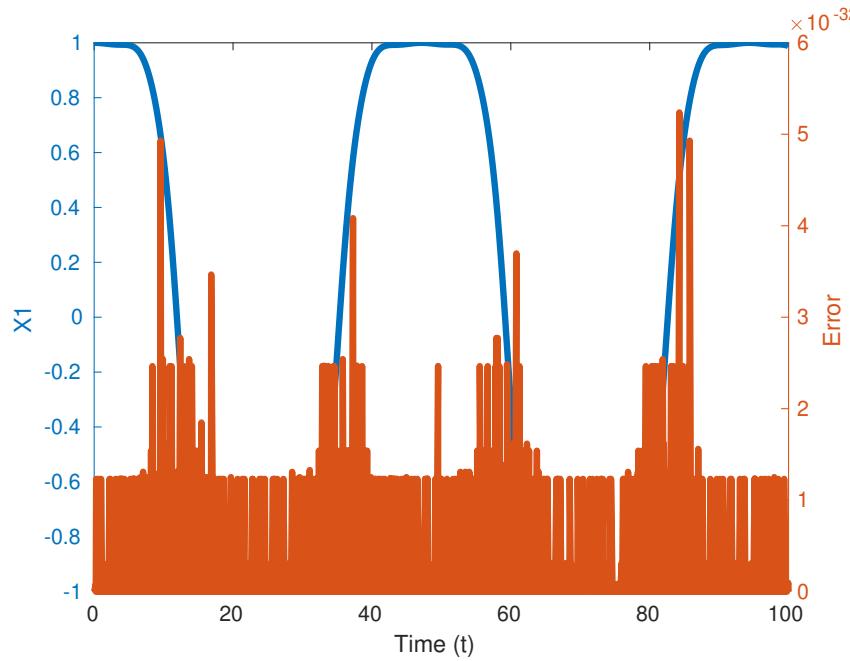
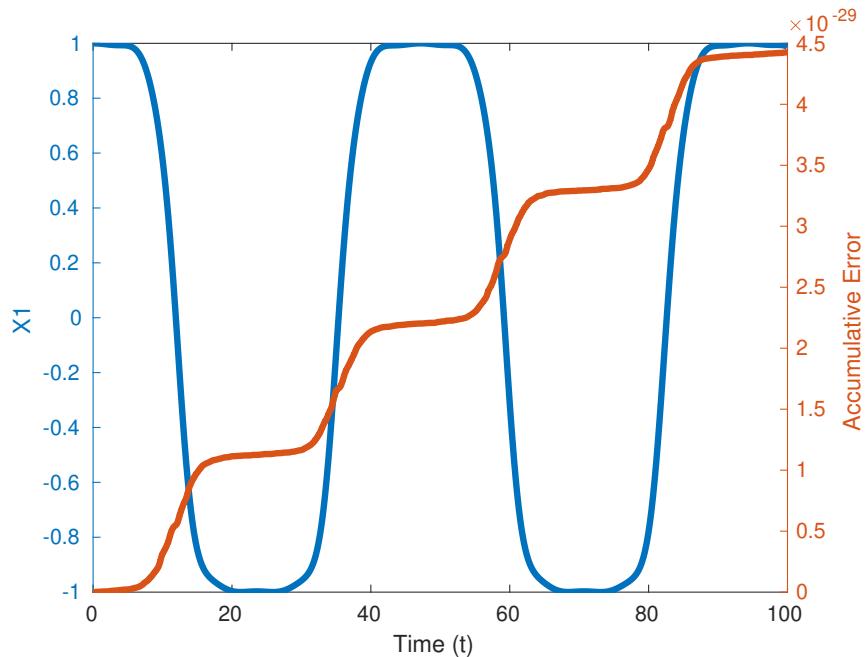
4.7.2 RK4, $h = 0.01$ Figur 11: Energien plottet mot tiden når $h = 0.01$ Figur 12: Feilen i X plottet mot tiden når $h = 0.01$

Figur 13: Feilen i X plottet mot energien når $h = 0.01$ Figur 14: Akkumulativ feil i X plottet mot energien når $h = 0.01$

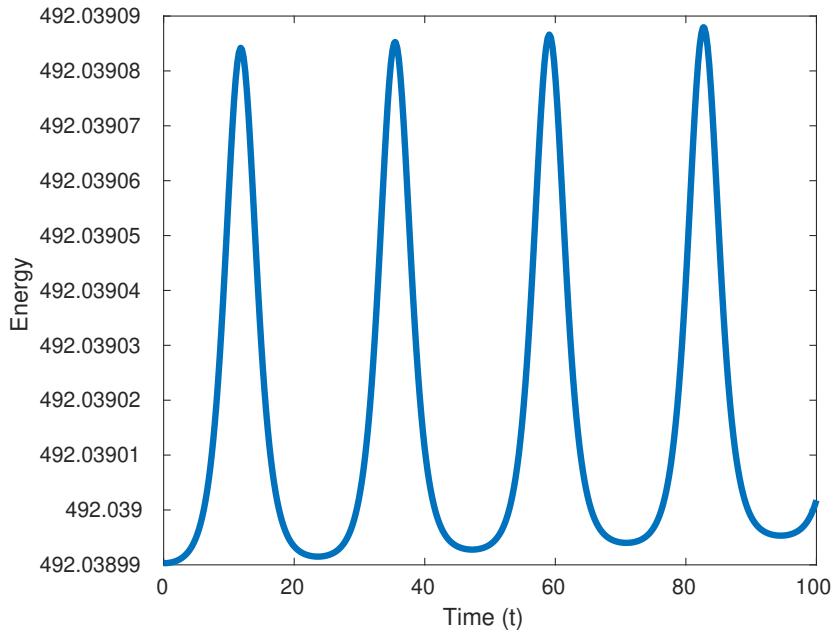
Figur 15: X_1 plottet mot feilen i X når $h = 0.01$ Figur 16: X_1 plottet mot akkumulativ feil i X når $h = 0.01$

4.7.3 RK4, $h = 0.001$ Figur 17: Energien plottet mot tiden når $h = 0.001$ Figur 18: Feilen i X plottet mot tiden når $h = 0.001$

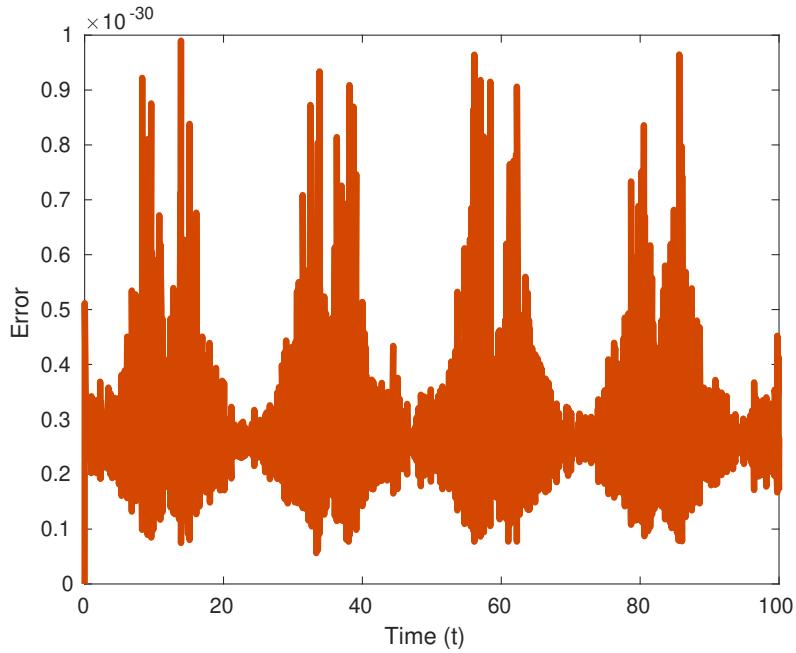
Figur 19: Feilen i X plottet mot energien når $h = 0.001$ Figur 20: Akkumulativ feil i X plottet mot energien når $h = 0.001$

Figur 21: X_1 plottet mot feilen i X når $h = 0.001$ Figur 22: X_1 plottet mot akkumulativ feil i X når $h = 0.001$

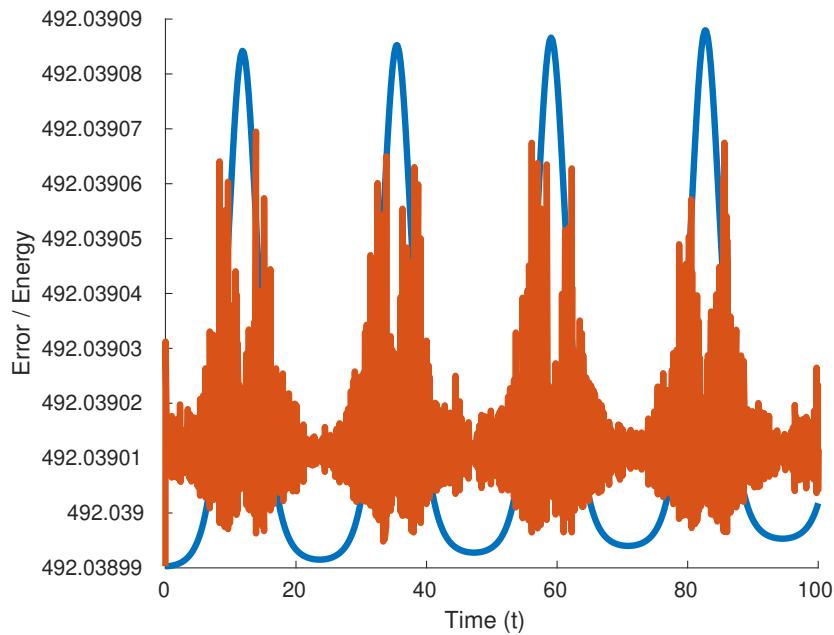
4.7.4 RKF45



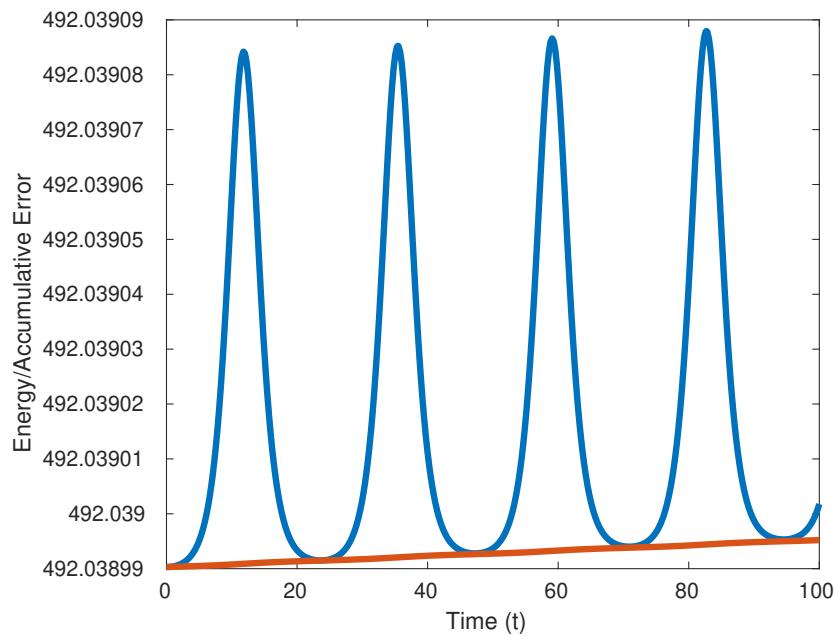
Figur 23: Energien plottet mot tiden for RKF45



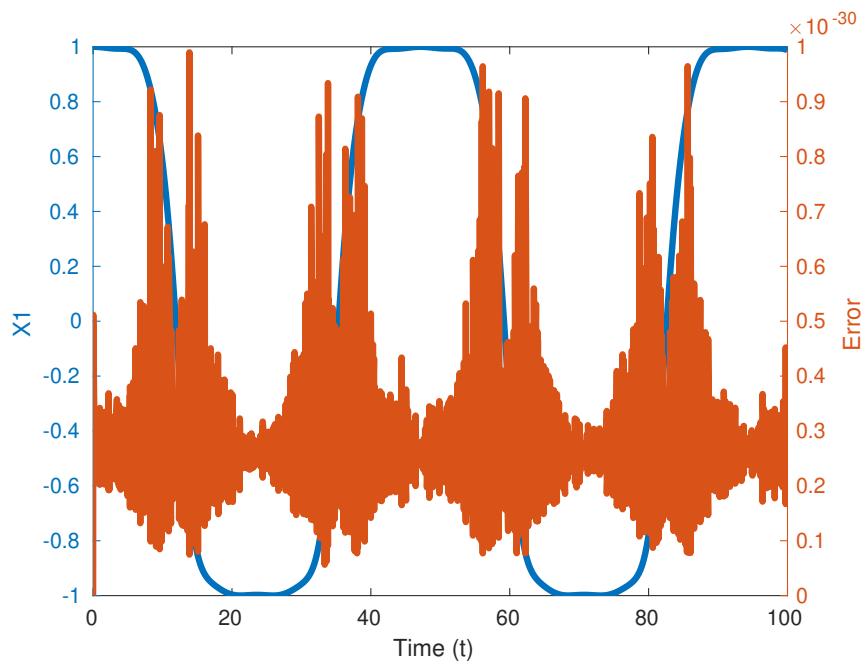
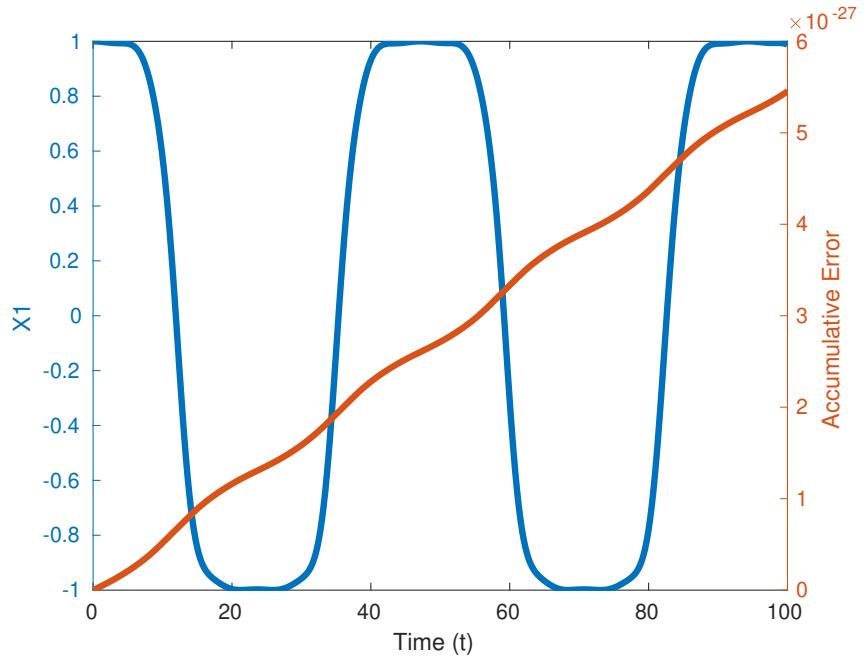
Figur 24: Feilen i X plottet mot tiden for RKF45

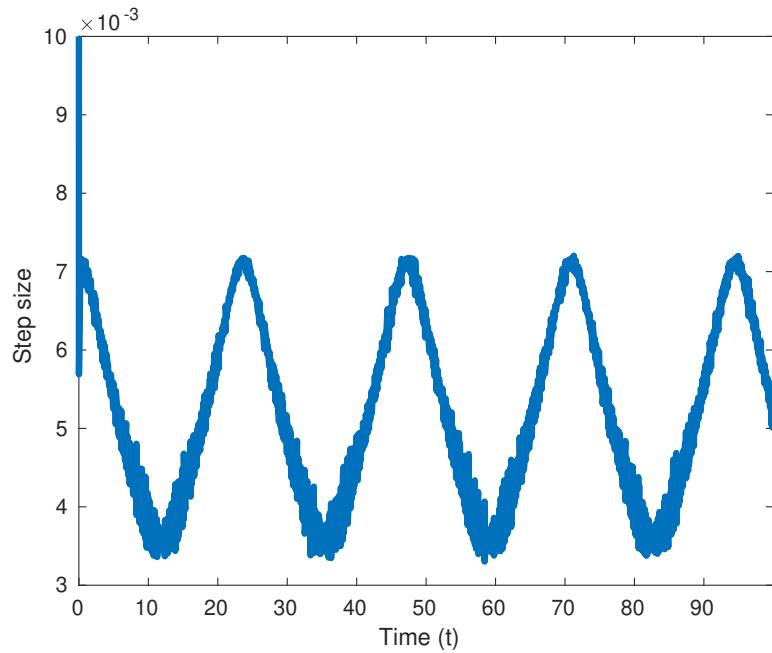


Figur 25: Feilen i X plottet mot energien for RKF45

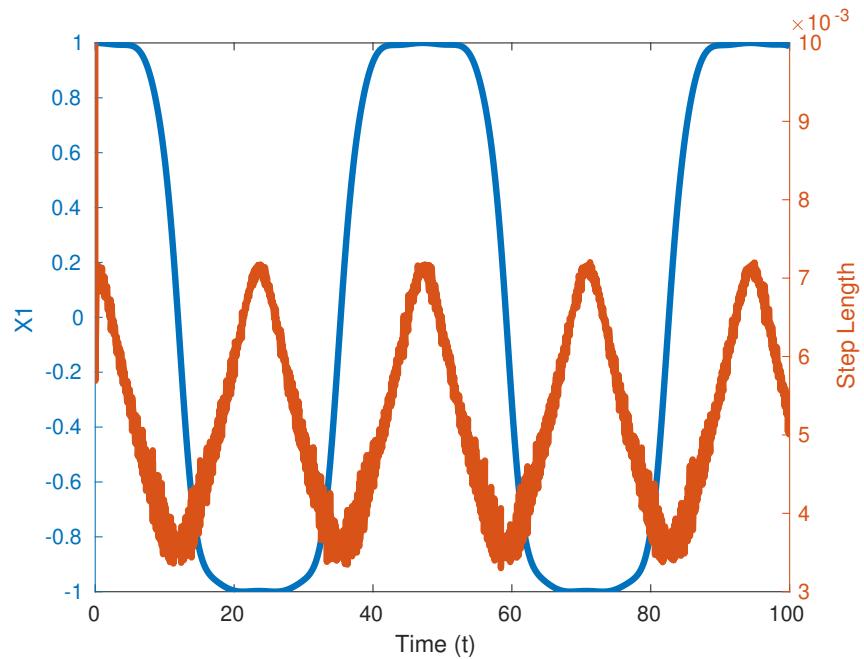


Figur 26: Akkumulativ feil i X plottet mot energien for RKF45

Figur 27: X_1 plottet mot feilen i X for RKF45Figur 28: X_1 plottet mot akkumulativ feil i X for RKF45



Figur 29: Steglengden plottet mot tiden for RKF45

Figur 30: Steglengden plottet X_1 for RKF45

Diskusjon

5.1 Rotasjon av kulelegemet

For rotasjon av kulelegemet fant vi at eksakt løsning var matrise (4.2.1). Som vi så ble en standard rotasjonsmatrise over 1. aksen. Dette er som forventet når vi snakker om en kule som roterer med et dreimoment over kun 1. aksen.

5.1.1 Numerisk feil

Problemstillingen med det roterende kulelegemet ble løst numerisk med newtons metode (oppgave 3), RK4 (oppgave 4) og RKF45 (oppgave 4). Den absolutte feilen for newtons metode, RK4 og RKF45 var henholdsvis i størrelsesordenen 10^{-9} , 10^{-12} og 10^{-13} . Dette viser at newtons metode er minst nøyaktig, og at RKF45 er mest nøyaktig, selv for dette trivielle tilfellet. Dette er i tråd med det vi forventet i forhold til teorien.

Energiforskjellen var lik for alle de tre numeriske metodene, nemlig 0. Dette stemmer med teorien. Energi kan ikke skapes eller forsvinne uten ytre påvirkning. I praksis betyr en energiforskjell på 0 at energiforskjellen er mindre enn maskinepsilon, som er 2.2204×10^{-16} . Generelt sett vil energiforskjellen være høyere for newtons metode på grunn av akkumulert numerisk feil. Dette ser vi ikke her fordi energiforskjellen er såpass liten at matlab runder den av til 0 for alle de tre numeriske metodene.

5.2 Rotasjon av T-nøkkelen

For rotasjon av t-nøkkelen utførte vi simuleringen med tre ulike rotasjonsvektorer. Disse tre rotasjonsvektorene tilsvarer rotasjon om enten 1., 2. eller 3. akse. For å få et klarere bilde av hva som skjer i hvert enkelt tilfelle, ser vi på hvordan komponentene til løsningen X varierer med tiden.

Merk at vi her referer til komponentene til X-matrisen med følgende indeksering:

$$X = \begin{bmatrix} X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 \\ X_7 & X_8 & X_9 \end{bmatrix}$$

5.2.1 Rotasjon om 1. akse

En rotasjon om 1. akse er i utgangspunktet være gitt ved matrisen:

$$R_1(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) \\ 0 & \sin(t) & \cos(t) \end{bmatrix}$$

Det vi da forventer er at X_5 og X_9 vil variere som cosinusbølger, mens X_6 og X_8 vil variere som sinusbølger. Dersom vi ser på figur 2, ser vi at dette stemmer. X_5 , X_6 , X_8 og X_9 oscillerer mellom 0 og 1, som forventet.

Så er det verdiene X_1 , X_2 , X_3 , X_4 og X_7 , som i utgangspunktet er henholdsvis 1, 0, 0, 0 og 0. Disse verdiene vil være konstante for en ren rotasjon om 1. aksen, men vi har rotasjonsvektor $\omega(0) = [1, 0.05, 0]$. Det er altså en liten rotasjon rundt 2. aksen også. Siden rotasjon om 2.aksen er såpass liten, forventer vi at verdiene vil være tilnærmet konstant 1, 0, 0, 0 og 0. Det ville de også vært, hadde det ikke vært for at treghetsmomentet til 1. aksen er den mellomliggende verdien. Da oppstår nemlig Dzhanibekov effekten.

Fra figur 2 kan vi lese av at:

$$X(0) \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X(20) \approx \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Det som her har skjedd, er at T-nøkkelen har snudd seg rundt. Det er dette som er Dzhani-bekov effekten, som er forklart i teoridelen. Effekten skjer også ved:

$$X(50) \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X(70) \approx \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X(90) \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

T-nøkkelen har altså snudd seg ved tidene $t = 20$, $t = 50$, $t = 70$ og $t = 90$.

5.2.2 Rotasjon om 2. akse

En rotasjon om 2. akse er i utgangspunktet være gitt ved matrisen:

$$R_2(t) = \begin{bmatrix} \cos(t) & 0 & \sin(t) \\ 0 & 1 & 0 \\ -\sin(t) & 0 & \cos(t) \end{bmatrix}$$

Det vi da forventer er at X_1 og X_9 vil variere som cosinusbølger, mens X_3 og X_7 vil variere som sinusbølger. Dersom vi ser på figur 3, ser vi at dette stemmer. X_1 , X_3 , X_7 og X_9 oscillerer mellom 0 og 1, som forventet.

Verdiene X_2 , X_4 , X_5 , X_6 og X_8 vil i utgangspunket være henholdsvis 0, 0, 1, 0 og 0. Siden rotasjonsvektoren er lik $\omega(0) = [0, 1, 0.05]$ forventer vi en liten rotasjon også rundt 3. aksen. Da vil X_2 , X_4 , X_5 , X_6 og X_8 variere litt i verdi. Dette ser vi stemmer, for i figur 3 er ikke verdiene rette streker, de er litt bølgete. Da er det varians i verdiene, som forventet. 2. aksen har ikke den mellomliggende verdien for trehetsmomentet, så her er det ingen Dzhanibekov effekt.

5.2.3 Rotasjon om 3. akse

En rotasjon om 3. akse er i utgangspunktet være gitt ved matrisen:

$$R_3(t) = \begin{bmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Det vi da forventer er at X_1 og X_5 vil variere som cosinusbølger, mens X_2 og X_4 vil variere som sinusbølger. Dersom vi ser på figur 4, ser vi at dette stemmer. X_1 , X_2 , X_4 og X_5 oscillerer mellom 0 og 1, som forventet.

Verdiene X_3, X_6, X_7, X_8 og X_9 vil i utgangspunktet være henholdsvis 0, 0, 0, 0 og 1. Siden rotasjonsvektoren er lik $\omega(0) = [0.05, 0, 1]$ forventer vi en liten rotasjon også rundt 1. aksen. Da vil X_3, X_6, X_7, X_8 og X_9 variere litt i verdi. Dette ser vi stemmer, for i figur 4 er ikke verdiene rette streker, de er litt bølgete. Da er det varians i verdiene, som forventet. 3. aksen har ikke den mellomliggende verdien for treghetsmomentet, så her er det ingen Dzhanibekov effekt.

5.2.4 Energien

Energien er bevart i alle tre tilfellene, for både RK4 og for RKF45. Den største endringen i energi er i størrelseordenen 10^{-5} , og den største feilen er 10^{-25} (hentet fra kapittel 4.5). Dermed kan vi anta at simuleringene av T-nøkkelen er korrekte. Mer om dette i neste kapittel.

5.2.5 Treghetsmomentet

Treghetsmomentet er gitt ved matrise (4.1.1). 1. akse har den mellomliggende verdien for treghetsmomentet. Som forventet er det rundt denne aksen Dzhanibekov effekten oppstår.

5.3 Numerisk feil

I denne seksjonen tolker vi den numeriske feilen i RK4 og RKF45. RK4 er kjørt med ulike steglengder, som beskrevet i kapittel 4.7. Vi ser kun på data for tilfellet hvor T-nøkkelen roterer rundt 1. aksen.

5.3.1 Energien

Energien skal være konstant. Det vil si at energien plottet opp mot tiden skal resultere i en rett strek. For figur 5, hvor $h = 0.1$, ser vi at dette ikke er tilfellet. Her er det to ting verdt å merke seg; energien endrer seg periodisk (øker og synker i gitte intervaller), samtidig som den har en generell økende trend over tid. Energien regnes ut v.h.a. løsningen X . For å forstå feilen i energien må vi derfor først se på feilen i løsningen X .

5.3.2 Feilen i X

Feilen i X over tid er gitt i figur 6, hvor $h = 0.1$. Her ser vi at den øker periodisk, men hvorfor er det slik? Figur 9 plotter feilen og den første komponenten i løsningen til X. Her er det en tydelig sammenheng; når X endrer seg rask ($\frac{dX}{dt}$ er stor), øker også feilen. Når X er stabil ($\frac{dX}{dt}$ er liten), er feilen liten. $\frac{dX}{dt}$ er stor hver gang T-nøkkelen vender om. T-nøkkelen vender om fire ganger, og feilen har derfor fire topper.

5.3.3 Sammenhengen mellom energien og feilen i X

Feilen i X plottet mot energien over tid er gitt i figur 7, hvor $h = 0.1$. Her er det ingen tvil; feilen i X påvirker energien. Dette kan tolkes som at dersom feilen i X er stor, blir også X litt for stor, som fører til at energien blir litt for stor. Dette forklarer hvorfor energien endrer seg periodisk, men hvorfor har den en økende trend over tid? Da må vi se på den akkumulative feilen i X.

5.3.4 Den akkumulative feilen i X

Den akkumulative feilen i X plottet mot X_1 over tid er gitt i figur 10, hvor $h = 0.1$. Her ser vi tydelig at når X_1 endrer seg raskt (T-nøkkelen vender om), så øker den akkumulative feilen. Når X_1 er stabil, er den akkumulative feilen stabil.

5.3.5 Sammenhengen mellom den akkumulative feilen i X og energien

Den akkumulative feilen i X plottet mot energien er gitt i figur 8, hvor $h = 0.1$. Her kommer det tydelig fram at den akkumulative feilen i X har ført til en akkumulativ feil i energien, slik at den får en økende trend.

Da er energien forklart. Formen den har, skyldes utelukkende av feilen i X, som igjen er avhengig av den fysiske bevegelsen til X. Merk at her er den første komponenten til X, X_1 brukt, men det er den totale forandringen i hele matrisen X som er avgjørende.

5.3.6 Steglengden i RK4

La oss sammenligne figur 5, hvor $h = 0.1$, med figur 11, hvor $h = 0.01$. Disse angir energien plottet mot tiden. Det er tydelig fra de to figurene at amplituden til energien er lavere for $h = 0.01$, som betyr at feilen er mindre. Det har heller ikke like stor økende trend som $h = 0.1$ har. Dersom vi ser på figur 17, hvor $h = 0.001$, ser vi at amplituden er enda mindre, og den økende trenden nesten er helt borte.

Hva sier dette oss? Som vi viste i de foregående delkapitlene er energien i praksis et mål på hvor stor feilen i X er. Det er ingen tvil om at lavere steglengde gir mer stabil energi, og dermed mindre feil. Dersom vi ser på feilen i X for de respektive steglengdene, ser vi den samme trenden. Jo lavere steglende, jo mindre feil.

5.3.7 Steglengden i RKF45

RKF45 varierer steglengden for å oppnå en gitt toleranse. Steglengden plottet mot X_1 er gitt i figur 30. Her er det en tydelig sammenheng; når X_1 endres raskt, senkes steglengden. Dette er for å imøtekommne den toleransen som er satt. Da skal også økningen i feil være relativt uavhengig av bevegelsen til T-nøkkelen.

Se figur 28. Det er X_1 plottet mot den akkumulative feilen. Her er det verdt å merke seg at den akkumulative feilen øker jevnt. Den er også ikke like avhengig av hvordan X_1 endrer seg. Likevel, dersom man ser på figur 24, som er feilen i X plottet mot tiden, er også denne periodisk. Samme med energien plottet mot tiden i figur 23.

Da er det enkelt å trekke konklusjonen om at variabel steglengde ikke har noe for seg. Energien er uansett periodisk, og vi kan oppnå samme nøyaktig med RK4 ved å ha tilstrekkelig liten steglengde. Algoritmisk sett er det likevel en forskjell. Der X er stabil, øker RKF45 steglengden for å spare prosesseringskraft. Det gjør ikke RK4.

Dermed kan faktisk RKF45 være raskere enn RK4 for problemer hvor X i hovedsak er stabil, men har noen periodiske raske endringer som krever lav steglengde. Videre arbeid kunne vært å ta tiden for diverse steglengder, og sammenligne for RK4 og RKF45.

Konklusjon

For T-nøkkelen ble Dzhanibekov effekten observert for en rotasjon over 1. aksen, men ikke for rotasjon over 2. eller 3. aksen. Dette stemmer med teorien, fordi det er 1. aksen som har den mellomliggende verdien for trehetsmomentet. Dzhanibekov effekten ble ikke observert for kulelegemet, fordi trehetsmomentet er likt for alle aksene.

Energien var bevart for rotasjon over 1., 2. og 3. aksen. Dette tilsier at simulasjonene våre er korrekte. Videre så vi på feilen, som var tilstrekkelig liten. Det viser seg også at det er en klar korrelasjon mellom feilen i energien og feilen i X.

For kulelegemet fant vi at newtons metode var minst nøyaktig, mens RKF45 var mest nøyaktig. For T-nøkkelen var RKF45 mer nøyaktig enn RK4, fordi RKF45 tilpasset seg bevegelsen til T-nøkkelen ved å variere steglengden. Ved å sette steglengden tilstrekkelig liten kunne vi oppnå samme nøyaktighet med RK4 som for RKF45, men (antageligvis) på bekostning av lengre prosesseringstid.

Referanser

- [1] H. J. Rivertz. (2020, okt) Rotasjon av stive legemer, prosjektoppgave i tdat3024. [Online]. Available: https://learn-eu-central-1-prod-fleet01-xythos.s3.eu-central-1.amazonaws.com/5def77a38a2f7/6919377?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27StiveLegemer%25287%2529.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20201028T060000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PL5M5HI5WH%2F20201028%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=432b30116a1c9db5200870d9f2350782866834d204ab74d9af51242b70099ba5
- [2] E. Weisstein. (oktober) Rotation matrix. (Lasted ned 29.10.2020). [Online]. Available: <https://mathworld.wolfram.com/RotationMatrix.html>
- [3] Veritasium. (2019, sep) The bizarre behavior of rotating bodies, explained. [Online]. Available: https://www.youtube.com/watch?v=1VPfZ_XzisU&t=280s&ab_channel=Veritasium
- [4] J. H. Mathews and K. K. Fink. (2020, okt) Runge-kutta-fehlberg method (rkf45). [Online]. Available: <http://maths.cnam.fr/IMG/pdf/RungeKuttaFehlbergProof.pdf>
- [5] T. Sauer, *Numerical Analysis*, 2nd ed. Pearson, 2012.
- [6] R. England. (2020, okt) Error estimates for runge-kutta type solutions to systems of ordinary differential equations. (Lasted ned 28.10.2020). [Online]. Available: https://www.researchgate.net/publication/31135539_Error_estimate_for_Runge-Kutta_type_solutions_to_systems_of_ordinary_differential_equations
- [7] T. Tao. (2019, nov) The “dzhanibekov effect” - an exercise in mechanics or fiction? explain mathematically a video from a space station. [Online]. Available: <https://mathoverflow.net/questions/81960/>

the-dzhanibekov-effect-an-exercise-in-mechanics-or-fiction-explain-mathemat

Vedlegg

Kildekoden er delt i to mapper, 'classes' og 'tasks'. Mappen 'classes' inneholder implementasjonen av de ulike metodene. Mappen 'tasks' bruker metodene fra 'classes' til å løse de respektive oppgavene.

Det er to typer matlab filer; vanlige matlab filer (.m) i 'classes', og live script filer (.mlx) i 'tasks'. Begynner med kildekoden i 'classes', deretter kildekoden i 'tasks'.

6.1 Kildekode

Program 1: Energy.m

```
%-----%
%                                         Energy.m
%
% Description:
%     Calculates the energy, K, for object
%
% Input:
%     X      SO(3), matrix in the Lie-group
%     I      Moment of inertia
%     w      Rotation vector
%
% Output:
%     K      Energy for object
%
%-----%

classdef Energy
    methods (Static)
        function K = calculate(L, w)
            K = (1/2) * dot(L, w);
        end
    end
end
```

Program 2: EulersMethod.m

```
%-----%
%                                         EulersMethod.m
%
% Description:
%     Class for eulers method
%
% Properties:
%     h      Step length
%     n      Number of iterations
%
%-----%
classdef EulersMethod

    properties
        h;
        n;
    end

    methods
        function obj = EulersMethod(h, n)
            obj.h = h;
            obj.n = n;
        end

        function [t, W] = solve(obj, X0, I, L)
            t = zeros(1, obj.n);
            W = cell(1, obj.n);

            t(1) = 0;
            W{1} = X0;

            for i = 1:obj.n
                o = Omega.toMatrix(I^(-1) * W{i}' * L);

                t(i+1) = t(i) + obj.h;
                W{i+1} = obj.step(W{i}, o);
            end
        end

        function Wout = step(obj, Win, o)
```

```

        Wout = Win * Exp.e(obj.h, o);
    end
end
end

```

Program 3: Exp.m

```

%-----%
%                                         Exp.m
%
% Description:
%     Calculates an orthogonal matrix with determinant = 1
%
% Input:
%     h      Step length
%     o      [ 0      -wZ      wY
%              wZ      0       -wX
%              -wY      wX      0   ]
%
% Output:
%     X      Orthogonal matrix
%
%-----%

classdef Exp
    methods (Static)

        % Calculates the exp
        function X = e(h, o)
            w = Omega.length(o);
            X = eye(3) + (1 - cos(w*h)) * (o^2/w^2) + sin(w*h) *
                (o/w);
        end

        % Checks X * X^T = I (within tolerance)
        function result = test(X, TOL)
            result = min(min(ismembertol(X * X.', eye(3), TOL)));
        end
    end
end

```

Program 4: Omega.m

```
%-----%
%                                         Omega.m
%
% Description:
%     Class used to convert between omega matrices and vectors
%
%-----%
classdef Omega
    methods(Static)

        % Converts omega matrix to omega vector
        function w = toVector(o)
            wX = o(3, 2); wY = o(1, 3); wZ = o(2, 1);
            w = [wX wY wZ]';
        end

        % Converts omega vector to omega matrix
        function o = toMatrix(w)
            o = [ 0      -w(3)   w(2)
                  w(3)      0      -w(1)
                  -w(2)    w(1)      0 ];
        end

        % Calculates length of omega matrix or vector
        function len = length(o)
            len = norm(o);
        end
    end
end
```

Program 5: RK4.m

```
%-----%
%                                         RK4.m
%
% Description:
%     Class for Runge-Kutta method
%
% Properties:
%     h      Step size
%     n      Number of iterations
%     TOL   Tolerance
%
% Properties (Constant):
%     A          First part of the Butcher tableau
%     B          Second part of the Butcher tableau
%
% Remark:
%     The Butcher tableau is not necessary to calculate the
%     approximate
%     solution in RK4. It is used for calculating the error.
%
%-----%
classdef RK4

    properties
        h;
        n;
    end

    properties(Constant)
        A = [ 0             0             0             0
              0             0             0             0
              1/4           0             0             0
              0             0             9/32          0
              3/32          -7200/2197  7296/2197   0
              0             0             -8            3680/513   -845/4104
              0             0             2             -3544/2565  1859/4104
              -8/27         0             0             0             ];
    end
```

```

        -11/40      0    ];
B = [ 25/216          0          1408/2565      2197/4104
      -1/5      0
      16/135      0          6656/12825      28561/56430
      -9/50    2/55 ];
end

methods
    function obj = RK4(h, n)
        obj.h = h;
        obj.n = n;
    end

    % Calculates the approximate solution
    function [t, W, E] = solve(obj, X0, I, L)

        % Initializes variables
        t = zeros(1, obj.n); t(1) = 0;
        E = zeros(1, obj.n); E(1) = 0;
        W = cell(1, obj.n); W{1} = X0;

        for i = 1:obj.n

            % Calculates sigma
            S1 = Sigma.toMatrix(I^(-1) * W{i}' * L);
            S2 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h/2, S1)
                * W{i}' * L);
            S3 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h/2, S2)
                * W{i}' * L);
            S4 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, S3)
                * W{i}' * L);

            % Calculates t
            t(i+1) = t(i) + obj.h;

            % Calculates W and Z
            W{i+1} = obj.wStep(W{i}, S1, S2, S3, S4);
            Z = obj.zStep(W{i}, I, L);

            % Calculates the error
            DW = W{i+1} - Z;
            E(i+1) = trace(transpose(DW) * DW);
        end
    end

```

```

    end

    % One step for calculating W
    function Wout = wStep(obj, Win, S1, S2, S3, S4)
        Wout = Win * Exp.e(obj.h/6, S1 + 2*S2 + 2*S3 + S4);
    end

    % One step for calculating Z
    function Zout = zStep(obj, Win, I, L)

        % Calculates sigma
        S1 = Sigma.toMatrix(I^(-1) * Win' * L);
        S2 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(2,
            1)*S1) * Win' * L);
        S3 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(3,
            1)*S1 + obj.A(3, 2)*S2) * Win' * L);
        S4 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(4,
            1)*S1 + obj.A(4, 2)*S2 + obj.A(4, 3)*S3) * Win' *
            L);
        S5 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(5,
            1)*S1 + obj.A(5, 2)*S2 + obj.A(5, 3)*S3 + obj.A(5,
            4)*S4) * Win' * L);
        S6 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(6,
            1)*S1 + obj.A(6, 2)*S2 + obj.A(6, 3)*S3 + obj.A(6,
            4)*S4 + obj.A(6, 5)*S5) * Win' * L);

        % Calculates Z
        Zout = Win * Exp.e(obj.h, obj.B(2, 1)*S1 + obj.B(2,
            2)*S2 + obj.B(2, 3)*S3 + obj.B(2, 4)*S4 + obj.B(2,
            5)*S5 + obj.B(2, 6)*S6);
    end
end
end

```

Program 6: RKF45.m

```

    0      0
    1/4      0
    0      0
    3/32      9/32
    0      0
    1932/2197 -7200/2197 7296/2197
    0      0
    439/216      -8
    0      0
    -8/27      2
    -11/40      0  ];
B = [ 25/216      0      1408/2565 2197/4104
    -1/5      0
    16/135      0      6656/12825 28561/56430
    -9/50      2/55 ];

MAX_ITERATIONS = 10;
end

methods
    function obj = RKF45(h, n, TOL)
        obj.h = h;
        obj.tEnd = h*n;
        obj.TOL = TOL;
    end

    % Calculates the approximate solution
    function [t, W, E, h] = solve(obj, X0, I, L)

        % Initializes variables
        W{1} = X0;
        t(1) = 0;
        E(1) = 0;
        h(1) = obj.h;

        i = 1;
        while(t(i) < obj.tEnd)

            % Calculates the approximate solution
            [t(i+1), W{i+1}, E(i+1)] = obj.step(t(i), W{i}, I
                , L);
            iterations = 0;
        end
    end
end

```

```

% Checks if the error is tolerable
if(E(i+1) > obj.TOL)
    obj = obj.adjustStep(E(i+1));
    [t(i+1), W{i+1}, E(i+1)] = obj.step(t(i), W{i
        }, I, L);
end

% Checks if the error is tolerable now
while(E(i+1) > obj.TOL)

    % Divides step size in two
    obj.h = obj.h / 2;

    % Recalculates the approximation
    [t(i+1), W{i+1}, E(i+1)] = obj.step(t(i), W{i
        }, I, L);

    % Checks if max iterations is exceeded
    iterations = iterations + 1;
    if(iterations > obj.MAX_ITERATIONS)
        error('MAX ITERATIONS EXCEEDED')
    end
end

% Forces the last t-value to be tEnd
if(t(i+1) > obj.tEnd)
    obj.h = obj.tEnd - t(i);
    [t(i+1), W{i+1}, E(i+1)] = obj.step(t(i), W{i
        }, I, L);
end

h(i+1) = obj.h;

% Adjusts the step for the next iteration
obj = obj.adjustStep(E(i+1));
i = i + 1;
end

h(end) = obj.h;
end

% One step in RKF45
function [tout, Wout, E] = step(obj, tin, Win, I, L)

```

```

% Calculates sigma
S1 = Sigma.toMatrix(I^(-1) * Win' * L);
S2 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(2,
    1)*S1) * Win' * L);
S3 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(3,
    1)*S1 + obj.A(3, 2)*S2) * Win' * L);
S4 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(4,
    1)*S1 + obj.A(4, 2)*S2 + obj.A(4, 3)*S3) * Win' *
L);
S5 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(5,
    1)*S1 + obj.A(5, 2)*S2 + obj.A(5, 3)*S3 + obj.A(5,
    4)*S4) * Win' * L);
S6 = Sigma.toMatrix(I^(-1) * Exp.e(-obj.h, obj.A(6,
    1)*S1 + obj.A(6, 2)*S2 + obj.A(6, 3)*S3 + obj.A(6,
    4)*S4 + obj.A(6, 5)*S5) * Win' * L);

% Calculates t
tout = tin + obj.h;

% Calculates W and Z
Wout = obj.wStep(Win, S1, S2, S3, S4, S5, S6);
Zout = obj.zStep(Win, S1, S2, S3, S4, S5, S6);

% Calculates the error
DW = Wout - Zout;
E = trace(transpose(DW) * DW);
end

% One step for calculating W
function Wout = wStep(obj, Win, S1, S2, S3, S4, S5, S6)
    Wout = Win * Exp.e(obj.h, obj.B(1, 1)*S1 + obj.B(1,
        2)*S2 + obj.B(1, 3)*S3 + obj.B(1, 4)*S4 + obj.B(1,
        5)*S5 + obj.B(1, 6)*S6);
end

% One step for calculating Z
function Zout = zStep(obj, Win, S1, S2, S3, S4, S5, S6)
    Zout = Win * Exp.e(obj.h, obj.B(2, 1)*S1 + obj.B(2,
        2)*S2 + obj.B(2, 3)*S3 + obj.B(2, 4)*S4 + obj.B(2,
        5)*S5 + obj.B(2, 6)*S6);
end

% Adjusts the step size to ensure tolerance is met
function obj = adjustStep(obj, E)

```

```

        if(E == 0)
            obj.h = 2 * obj.h;
        else
            obj.h = 0.8 * (obj.TOL / E)^(1/6) * obj.h;
        end

        if(obj.h == inf)
            disp(['Error', num2str(E)])
        end
    end
end

```

Program 7: Sigma.m

```

%-----%
%                               Sigma.m
%
% Description:
%     Class used to convert between sigma matrices and vectors
%
%-----

classdef Sigma
    methods(Static)

        % Converts sigma vector to sigma matrix
        function S = toMatrix(s)
            S = [ 0      -s(3)   s(2)
                  s(3)    0      -s(1)
                  -s(2)   s(1)    0 ];
        end
    end
end

```

Program 8: THandle.m

```
%-----%
%                                         THandle.m
%
% Description:
%     Class for T-Handle
%
% Properties:
%     R1      Radius of cylinder 1
%     R2      Radius of cylinder 2
%     L1      Length og cylinder 1
%     L2      Length og cylinder 2
%     M       Mass of T-Handle
%     p       Mass density of cylinders
%     I       Moment of inertia matrix
%
%-----%
classdef THandle
properties
    R1; R2;
    L1; L2;
    M;
    P;
    I;
end

methods
    function obj = THandle(r1, r2, l1, l2, m, p)
        obj.R1 = r1; obj.R2 = r2;
        obj.L1 = l1; obj.L2 = l2;
        obj.M = m;
        obj.P = p;
    end

    % Calculates the moment of inertia
    function [I, obj] = calculateMomentOfInertia(obj)
        TOL = 1e-12;

        % Calculates masses
        M1 = pi * obj.R1^2 * obj.L1 * obj.P;
        M2 = pi * obj.R2^2 * obj.L2 * obj.P;
    end

```

```
% Verifies that the masses add up to the total mass
if (M1 + M2) - obj.M > TOL
    error('Invalid input for masses! M1 + M2 ~= M');
end

% Calculates components of I
Ixx = (M1 * obj.R1^2)/4 + (M1 * obj.L1^2)/12 + (M2 *
obj.R2^2)/2;
Iyy = (M1 * obj.R1^2) + (M2 * obj.L2^2)/4 + (M1 * obj
.R1^2)/2 + (M2 * obj.R2^2)/4 + (M2 * obj.L2^2)/12;
Izz = (M1 * obj.R1^2) + (M2 * obj.L2^2)/4 + (M1 * obj
.R1^2)/4 + (M1 * obj.L1^2)/12 + (M2 * obj.R2^2)/4
+ (M2 * obj.L2^2)/12;

% Creates moment of inertia matrix
I = diag([Ixx Iyy Izz]);
obj.I = I;
end
end
end
```

Program 9: Visualization.m

```
%-----%
%                               Visualization.m
%
% Description:
%     Class that visualizes (plots and animates) the T-Handle
%
% Properties:
%     tHandle      The T-Handle to be animated
%     output       The current output (used for defining output
%                  path)
%
%-----%
%
classdef Visualization
    properties
        tHandle;
        output;
```

```

end

methods
    function obj = Visualization(tHandle, output)
        obj.tHandle = tHandle;
        obj.output = output;

        % Creates output folder for figures
        path = [pwd, '/figures/', output, '/f_components'];
        if ~isfolder(path)
            mkdir(path);
        end

        % Creates output folder for animations
        path = [pwd, '/animations/', output, '/'];
        if ~isfolder(path)
            mkdir(path);
        end
    end

    % Plots all the X-components in a grid
    function plotX(obj, t, X)
        path = [pwd, '/figures/', obj.output, '/'];

        for i = 1:9

            % The x component
            Xc = cellfun(@(x) x(i), X);

            % Creates plot for the x-component
            f = figure('visible', 'off');
            plot(t, Xc, 'LineWidth', 3);
            xlim([0, t(end)]); ylim([-1, 1]);
            xlabel('Time (t)'); ylabel(['X', num2str(i)]);

            % Saves plot as png
            print([path, 'f_components/F', num2str(i)], '-dpng');

            close(f);
        end

        % Creates image collage from each plot
        image = [imread([path, 'f_components/F1.png']) imread

```

```

([path, 'f_components/F2.png']) imread([path, 'f_components/F3.png']); ...
imread([path, 'f_components/F4.png']) imread
([path, 'f_components/F5.png']) imread([
path, 'f_components/F6.png']); ...
imread([path, 'f_components/F7.png']) imread
([path, 'f_components/F8.png']) imread([
path, 'f_components/F9.png)]);

imwrite(image, [path, 'F.png']); % Saves image
end

% Plots the step-size diagram
function plotH(obj, t, h)
    path = [pwd, '/figures/', obj.output, '/'];

    f = figure('visible', 'off');
    plot(t, h, 'LineWidth', 3);
    xlim([0, t(end - 1)]);
    xlabel('Time (t)'); ylabel('Step size');

    % Saves plot as .epsc
    print([path, 'h'], '-depsc');

    close(f);
end

% Plots the step-size diagram
function plotHX(obj, t, h, X, c)
    path = [pwd, '/figures/', obj.output, '/'];

    % The x component
    Xc = cellfun(@(x) x(c), X);

    f = figure('visible', 'off');

    xlim([0, t(end)]);
    xlabel('Time (t)');

    % Plots Xc
    yyaxis left
    plot(t, Xc, 'LineWidth', 3);
    ylabel(['X', num2str(c)]);

```

```

% Plots h
yyaxis right
plot(t, h, 'LineWidth', 3);
ylabel('Step Length');

% Saves plot as .epsc
print([path, 'HX', num2str(c)], '-depsc');

close(f);
end

% Plots the energy diagram
function plotEnergy(obj, t, energy)
    path = [pwd, '/figures/', obj.output, '/'];

    f = figure('visible', 'off');
    plot(t, energy, 'LineWidth', 3);
    xlim([0, t(end)]);
    xlabel('Time (t)'); ylabel('Energy');

    % Saves plot as .epsc
    print([path, 'Energy'], '-depsc');

    close(f);
end

% Plots the error diagram
function plotError(obj, t, E)
    path = [pwd, '/figures/', obj.output, '/'];

    f = figure('visible', 'off');
    plot(t, E, 'LineWidth', 3, 'Color', '#d44700');
    xlim([0, t(end)]);
    xlabel('Time (t)'); ylabel('Error');

    % Saves plot as .epsc
    print([path, 'Error'], '-depsc');

    close(f);
end

% Plots the X-Error diagram
function plotErrorX(obj, t, X, c, E)
    path = [pwd, '/figures/', obj.output, '/'];

```

```

% Calculates the x component
Xc = cellfun(@(x) x(c), X);

f = figure('visible', 'off');

xlim([0, t(end)]);
xlabel('Time (t)');

% Plots Xc
yyaxis left
plot(t, Xc, 'LineWidth', 3);
ylabel(['X', num2str(c)]);

% Plots error
yyaxis right
plot(t, E, 'LineWidth', 3);
ylabel('Error');

% Saves plot as .epsc
print([path, 'ErrorX1'], '-depsc');

close(f);
end

% Plots the Energy-Error diagram
function plotErrorEnergy(obj, t, energy, E, c)
    path = [pwd, '/figures/', obj.output, '/'];

    f = figure('visible', 'off');

    hold on

    plot(t, energy, 'LineWidth', 3);
    plot(t, (E * c) + energy(1), 'LineWidth', 3);

    xlim([0, t(end)]);
    xlabel('Time (t)');
    ylabel('Error / Energy');

    hold off

    % Saves plot as .epsc
    print([path, 'ErrorEnergy'], '-depsc');

```

```
        close(f);
    end

    % Plots the accumulative error diagram
    function plotAccumulativeError(obj, t, E)
        path = [pwd, '/figures/', obj.output, '/'];

        % Calculates accumulative error
        for i = 2:length(E)
            E(i) = E(i) + E(i-1);
        end

        f = figure('visible', 'off');
        plot(t, E, 'LineWidth', 3, 'Color', '#d44700');
        xlim([0, t(end)]);
        xlabel('Time (t)'), ylabel('Accumulative Error');

        % Saves plot as .epsc
        print([path, 'AccumulativeError'], '-depsc');

        close(f);
    end

    % Plots the accumulative error diagram
    function plotAccumulativeErrorX(obj, t, X, c, E)
        path = [pwd, '/figures/', obj.output, '/'];

        % The x component
        Xc = cellfun(@(x) x(c), X);

        % Calculates accumulative error
        for i = 2:length(E)
            E(i) = E(i) + E(i-1);
        end

        f = figure('visible', 'off');

        xlim([0, t(end)]);
        xlabel('Time (t)');

        % Plots Xc
        yyaxis left
        plot(t, Xc, 'LineWidth', 3);
```

```

    ylabel(['X', num2str(c)]);

    % Plots accumulative error
    yyaxis right
    plot(t, E, 'LineWidth', 3);
    ylabel('Accumulative Error');

    % Saves plot as .epsc
    print([path, 'AccumulativeErrorX1'], '-depsc');

    close(f);
end

% Plots the accumulative error diagram
function plotAccumulativeErrorEnergy(obj, t, energy, E, c)
    path = [pwd, '/figures/', obj.output, '/'];

    % Calculates accumulative error
    E(1) = E(1);
    for i = 2:length(E)
        E(i) = E(i) + E(i-1);
    end

    E = E * c;
    E = E + energy(1);

    f = figure('visible', 'off');

    % Plots energy
    hold on
    xlim([0, t(end)]);
    xlabel('Time (t)');
    ylabel('Energy/Accumulative Error');

    plot(t, energy, 'LineWidth', 3);
    plot(t, E, 'LineWidth', 3);

    box on
    hold off

    % Saves plot as .epsc
    print([path, 'AccumulativeErrorEnergy'], '-depsc');

```

```

        close(f);
    end

    % Animates the spinning T-Handle
    function animateTHandle(obj, X, h)
        path = [pwd, '/animations/', obj.output];

        % Length of cylinders
        L1 = obj.tHandle.L1;
        L2 = obj.tHandle.L2;

        % Used for indexing
        x = 1; y = 2; z = 3;

        % Point A
        A = [ -1      % x
              0      % y
              0 ]; % z

        % Pre-allocation for speed
        B = cell(1, length(X));
        C = cell(1, length(X));
        D = cell(1, length(X));

        % Updates points B, C and D in relation to X
        for i = 1:length(X)
            e1 = X{i}(:, 1);
            e2 = X{i}(:, 2);
            e3 = X{i}(:, 3);

            B{i} = A + L1/2 * e2;
            C{i} = A - L1/2 * e2;
            D{i} = L2 * e1;
        end

        % Sets up figure window
        figure('visible', 'off');
        plot3(0, 0, 0);

        xlabel('X (cm)');
        ylabel('Y (cm)');
        zlabel('Z (cm)');

        axis equal;
    end

```

```

        xlim(L1*[-1, 1]);
        ylim(L1*[-1, 1]);
        zlim(L2*[-1, 1]);

        grid on;

% Draws the T-Handle
AD = line('xdata', [A(x), D{1}(x)], 'ydata', [A(y)
    , D{1}(y)], 'zdata', [A(z), D{1}(z)], 'color
    ', 'k', 'linewidth', 5);
BC = line('xdata', [B{1}(x), C{1}(x)], 'ydata', [B
    {1}(y), C{1}(y)], 'zdata', [B{1}(z), C{1}(z)], 'color
    ', 'k', 'linewidth', 5);

% Starts animation
animation = VideoWriter([path, '/THandle'], 'MPEG-4')
    ;
animation.FrameRate = 1/h;
open(animation);

n = 0;
for i = 1:length(X)

    % Redraws T-Handle in new orientation
    set(AD, 'xdata', [A(x), D{i}(x)], 'ydata', [A(
        y), D{i}(y)], 'zdata', [A(z), D{i}(z)]);
    set(BC, 'xdata', [B{i}(x), C{i}(x)], 'ydata', [B{
        i}(y), C{i}(y)], 'zdata', [B{i}(z), C{i}(z)]);

    % Writes frame to video
    writeVideo(animation, getframe(gcf));
end

% Stops animation
close(animation);
end
end

```

Task 1

Defines the file path

```
addpath('classes');
```

Task a

Task description

Implementer eksponentfunksjonen i likning (21). Input skal være en matrise på formen (18). Funksjonen bør ta inn både Ω og h som input. Lag et testprogram som sjekker om output X er en matrise som oppfyller kravet $X^T X = I$

Parameter initialization

```
% Omega matrix
o = [ 0   -3    2
      3    0   -1
     -2   1    0 ];

h = 0.1;           % Step size
X = Exp.e(h, o) % X-Matrix
```

X = 3x3

0.9358	-0.2832	0.2102
0.3029	0.9506	-0.0680
-0.1805	0.1273	0.9753

Tests the Exp-function

```
TOL = 1e-12;
Exp.test(X, TOL)
```

ans = logical
1

Answer is 1, which is equal to true. In other words, the matrix X fulfills the requirement of $X^T X = I$ within a tolerance of 1e-12.

Task b

Task description

Implementer en funksjon som regner ut energien til et roterende legeme som har treghetsmoment/og rotasjonsvektor $\vec{\omega}$. Denne vil dere trenge til å sjekke om simuleringene bevarer energien til T-nøkkelen.

Parameter initialization

```
w = [1, 0, 0]'; % Rotation vector
L = [1, 0, 0]'; % Torque vector
```

Calculates the energy

```
K = Energy.calculate(L, w)
```

K = 0.5000

Total energy for this system is 0.5.

Task c

Task description

Regn ut treghetsmomentet til T-nøkkelen ved å bruke formelen i forrige avsnitt.

Parameter initialization

```
R1 = 1; R2 = 1; % Radius
L1 = 8; L2 = 4; % Length
p = 6.7;          % Mass density
M = 12*pi * p;  % Mass
```

Calculates the moment of inertia

```
tHandle = THandle(R1, R2, L1, L2, M, p);
I = tHandle.calculateMomentOfInertia()
```

```
I = 3x3
10^3 ×
0.9823      0      0
0      0.7227    0
0      0      1.5787
```

The moment of inertia for the T-Handle is given above.

Task 2

Defines file path

```
addpath('classes');
```

Task description

Gitt spesialtilfellet der legemet er en kule. Dere kan da anta at treghetsmomentet er lik identitetsmatrisen. La også

startbetingeslen til $X(t)$ være lik identitetsmatrisen. La $L = (1, 0, 0)$ Løs likningene (19) og (20) eksakt.

Task solution

```
X = @ (x) [ 1      0      0  
            0 cos(x) -sin(x)  
            0 sin(x)  cos(x) ];
```

The exact solution is equal to the rotation matrix about the X-axis. The details are given in the report.

Task 3

Defines file path

```
addpath('classes');
```

Task description

Implementer varianten av Eulers metode gitt i likning (25) i avsnitt 4.1. Og test metoden på systemet som består av likningene (19) og (20).

Parameter initialization

```
X0 = eye(3); % X-matrix
I = eye(3); % Moment of inertia matrix
L = [1 0 0]'; % Torque vector

h = 0.1; % Step size
n = 10000; % Number of iterations
```

Approximates the solution with Eulers method

```
eulersMethod = EulersMethod(h, n);
[t, W] = eulersMethod.solve(X0, I, L);
W = W{end}

W = 3x3
    1.0000      0      0
    0    0.5624   -0.8269
    0    0.8269    0.5624
```

The exact solution

```
X = @ (x) [ 1      0          0
            0    cos(x) -sin(x)
            0    sin(x)  cos(x) ];

X = X(t(end))

X = 3x3
    1.0000      0      0
    0    0.5624   -0.8269
    0    0.8269    0.5624
```

Calculates the error

```
error = abs(X - W)

error = 3x3
10^-9 x
      0      0      0
      0  0.1309  0.0897
      0  0.0897  0.1309
```

The above output shows that there is an absolute error of 1e-9 for the approximate solution.

Checks if energy is conserved

```
w0 = (X0 * I)^(-1) * L;
E0 = Energy.calculate(L, X0 * w0);

w1 = (W * I)^(-1) * L;
E1 = Energy.calculate(L, W * w1);

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', E0, E1, abs(E1 - E0))

Initial energy: 5.000000e-01
Final energy: 5.000000e-01
Difference: 0
```

Task 4

Defines file path

```
addpath('classes');
```

Task description

Implementer varianten av RK4-metoden beskrevet i avsnitt 4.4. Dere kan om dere vil implementere varianten av Runge Kutta Fehlberg-metoden (RKF45) som er beskrevet i avsnitt 4.5. Vær nøyne på å implementere metoden nøyaktig slik den står. En liten feil i koefisientene vil gjøre metoden omtrent like unøyaktig som Eulers metode.

Parameter initialization

```
X0 = eye(3); % X-matrix
I = eye(3); % Moment of inertia matrix
L = [1 0 0]'; % Torque vector

h = 0.1; % Step size
n = 10000; % Number of iterations
TOL = 1e-50; % Tolerance
```

RKF45

```
rkf45 = RKF45(h, n, TOL);
[t, W] = rkf45.solve(X0, I, L);
W45 = W{end}

W45 = 3x3
1.0000      0      0
0      0.5624   -0.8269
0      0.8269    0.5624
```

RK4

```
rk4 = RK4(h, n);
[~, W] = rk4.solve(X0, I, L);
W4 = W{end}

W4 = 3x3
1.0000      0      0
0      0.5624   -0.8269
0      0.8269    0.5624
```

The exact solution

```
X = @(x) [ 1      0      0
            0  cos(x) -sin(x)
```

```

          0 sin(x) cos(x) ];

X = X(t(end))

X = 3x3
 1.0000      0      0
 0    0.5624   -0.8269
 0    0.8269    0.5624

```

Calculates the error

RK4

```
RK4Error = abs(X - W4)
```

```

RK4Error = 3x3
10^-12 x
 0      0      0
 0    0.3871   0.4192
 0    0.4192   0.3871

```

RKF45

```
RKF45Error = abs(X - W45)
```

```

RKF45Error = 3x3
10^-13 x
 0      0      0
 0    0.1887   0.1288
 0    0.1288   0.1887

```

We can see from this that RK4 and RKF45 are more accurate than newtons method, even on this trivial example.

Checks if energy is conserved

```
w0 = (X0 * I)^(-1) * L;
E0 = Energy.calculate(L, X0 * w0);
```

RK4

```

w1 = (W4 * I)^(-1) * L;
E1 = Energy.calculate(L, W4 * w1);

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', E0, E1, abs(E1 - E0))

```

```

Initial energy: 5.000000e-01
Final energy: 5.000000e-01

```

```
Difference:      0
```

RKF45

```
w1 = (W45 * I)^(-1) * L;
E1 = Energy.calculate(L, W45 * w1);

fprintf('Initial energy: %i \nFinal energy:   %i \nDifference:      %i', E0, E1, abs(E1 - E0))
```

```
Initial energy: 5.000000e-01
Final energy:   5.000000e-01
Difference:      0
```

Task 5-6

Defines file path

```
addpath('classes');
```

Task description

Bruk implementeringen av RK4 eller RKF45 til å løse systemet. Benytt treghetsmomentent til T-nøkkelen som ble utregnet i tidligere oppgave. Bruk $X(0) = I$ (3×3 identitetsmatrisen). Beregn L slik at $\omega(0)$ har lengde ca lik 1 ved tiden 0. Kjør eksperimentet med:

- a) $\omega(0) = [1, 0.05, 0]^T$
- b) $\omega(0) = [0, 1, 0.05]^T$
- c) $\omega(0) = [0.05, 0, 1]^T$

Steglengden lar dere være så liten at energien ikke endrer seg nevneverdig

Tegn opp komponentene til løsningene X fra oppgave 5 som ni funksjoner av tiden. Gi en tolkning av disse grafene. Alternativt kan dere lage en 3-D animasjon av T-nøkkelen som roterer

Parameter initialization

```
R1 = 1; R2 = 1; % Radius
L1 = 8; L2 = 4; % Length
p = 6.7;          % Mass density
M = 12*pi * p;  % Mass

X0 = eye(3); % X-matrix

h = 0.01;      % Initial step length
n = 10000;     % Number of iterations
TOL = 1e-30;   % Tolerance
```

Calculates moment of inertia

```
tHandle = THandle(R1, R2, L1, L2, M, p);
I = tHandle.calculateMomentOfInertia();
```

Task a

Parameter initialization

```
w0 = [1, 0.05, 0]'; % Rotation vector
L = X0 * I * w0; % Torque vector
```

RKF45

Approximates solution

```
rkf45 = RKF45(h, n, TOL);
[t, W, E, h] = rkf45.solve(X0, I, L);
fprintf('Max Error: %d', max(E));
```

Max Error: 9.795761e-31

Calculates energy

```
energy = zeros(length(W), 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener
```

Initial energy: 4.920390e+02
Final energy: 4.920390e+02
Difference: 1.149231e-05

Visualization

```
v = Visualization(tHandle, '6a/RKF45');
v.plotX(t, W);
v.plotH(t, h);
v.plotHX(t, h, W, 1);
v.plotError(t, E);
v.plotErrorX(t, W, 1, E);
v.plotEnergy(t, energy);
v.plotErrorEnergy(t, energy, E, 8.0*10^(25));
v.plotAccumulativeError(t, E);
v.plotAccumulativeErrorX(t, W, 1, E);
v.plotAccumulativeErrorEnergy(t, energy, E, 9*10^(20));
v.animateTHandle(W, t(end) / length(W));
```

RK4, h = 0.1

Approximates solution

```

h = 0.1; n = 1000;
rk4 = RK4(h, n);
[t, W, E] = rk4.solve(X0, I, L);
fprintf('Max Error: %d', max(E));

```

Max Error: 5.597257e-15

Calculates energy

```

energy = zeros(n, 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener

```

Initial energy: 4.920390e+02
Final energy: 4.921088e+02
Difference: 6.982597e-02

Visualization

```

v = Visualization(tHandle, '6a/RK4/0.1');
v.plotX(t, W);
v.plotError(t, E);
v.plotErrorX(t, W, 1, E);
v.plotEnergy(t, energy);
v.plotErrorEnergy(t, energy, E, 1.15*10^(13));
v.plotAccumulativeError(t, E);
v.plotAccumulativeErrorX(t, W, 1, E);
v.plotAccumulativeErrorEnergy(t, energy, E, 5.7*10^(10));

```

RK4, h = 0.01**Approximates solution**

```

h = 0.01; n = 10000;
rk4 = RK4(h, n);
[t, W, E] = rk4.solve(X0, I, L);
fprintf('Max Error: %d', max(E));

```

Max Error: 5.567912e-25

Calculates energy

```

energy = zeros(n, 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener

```

Initial energy: 4.920390e+02
Final energy: 4.920391e+02
Difference: 8.296730e-05

Visualization

```

v = Visualization(tHandle, '6a/RK4/0.01');
v.plotX(t, W);
v.plotError(t, E);
v.plotErrorX(t, W, 1, E);
v.plotEnergy(t, energy);
v.plotErrorEnergy(t, energy, E, 1.0*10^(21));
v.plotAccumulativeError(t, E);
v.plotAccumulativeErrorX(t, W, 1, E);
v.plotAccumulativeErrorEnergy(t, energy, E, 5.5*10^(16));

```

RK4, h = 0.001

Approximates solution

```

h = 0.001; n = 100000;
rk4 = RK4(h, n);
[t, W, E] = rk4.solve(X0, I, L);
fprintf('Max Error: %d', max(E));

```

Max Error: 4.949640e-32

Calculates energy

```

energy = zeros(n, 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener

```

Initial energy: 4.920390e+02
Final energy: 4.920390e+02

```
Difference: 2.603802e-07
```

Visualization

```
v = Visualization(tHandle, '6a/RK4/0.001');
v.plotX(t, W);
v.plotError(t, E);
v.plotErrorX(t, W, 1, E);
v.plotEnergy(t, energy);
v.plotErrorEnergy(t, energy, E, 5.0*10^(25));
v.plotAccumulativeError(t, E);
v.plotAccumulativeErrorX(t, W, 1, E);
v.plotAccumulativeErrorEnergy(t, energy, E, 1.1*10^(21));
```

Task b

Parameter initialization

```
h = 0.01; n = 10000;
w0 = [0, 1, 0.05]'; % Rotation vector
L = X0 * I * w0; % Torque vector
```

RKF45

Approximates solution

```
rkf45 = RKF45(h, n, TOL);
[t, W, E, h] = rkf45.solve(X0, I, L);
fprintf('Max Error: %d', max(E));
```

```
Max Error: 8.369184e-31
```

Calculates energy

```
energy = zeros(length(W), 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener
```

```
Initial energy: 3.633088e+02
Final energy: 3.633088e+02
Difference: 8.599500e-07
```

Visualization

```
v = Visualization(tHandle, '6b/RKF45');
v.plotX(t, W);
v.plotH(t, h);
v.plotError(t, E);
v.plotEnergy(t, energy);
v.plotAccumulativeError(t, E);
v.animateTHandle(W, t(end) / length(W));
```

RK4

Approximates solution

```
h = 0.01;
rk4 = RK4(h, n);
[t, W, E] = rk4.solve(X0, I, L);
fprintf('Max Error: %d', max(E));
```

Max Error: 3.716739e-28

Calculates energy

```
energy = zeros(n, 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener
Initial energy: 3.633088e+02
Final energy: 3.633088e+02
Difference: 1.697041e-06
```

Visualization

```
v = Visualization(tHandle, '6b/RK4');
v.plotX(t, W);
v.plotError(t, E);
v.plotEnergy(t, energy);
v.plotAccumulativeError(t, E);
```

Task c

Parameter initialization

```
w0 = [0.05, 0, 1]'; % Rotation vector
L = X0 * I * w0;      % Torque vector
```

RKF45

Approximates solution

```
rkf45 = RKF45(h, n, TOL);
[t, W, E, h] = rkf45.solve(X0, I, L);
fprintf('Max Error: %d', max(E));
```

Max Error: 5.828470e-31

Calculates energy

```
energy = zeros(length(W), 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener
```

Initial energy: 7.905530e+02
 Final energy: 7.905530e+02
 Difference: 3.418527e-07

Visualization

```
v = Visualization(tHandle, '6c/RKF45');
v.plotX(t, W);
v.plotH(t, h);
v.plotError(t, E);
v.plotEnergy(t, energy);
v.plotAccumulativeError(t, E);
v.animateTHandle(W, t(end) / length(W));
```

RK4

Approximates solution

```
h = 0.01;
```

```
rk4 = RK4(h, n);
[t, W, E] = rk4.solve(x0, I, L);
fprintf('Max Error: %d', max(E));
```

Max Error: 3.018502e-25

Calculates energy

```
energy = zeros(n, 1);

for i = 1:length(W)
    w = (W{i} * I)^(-1) * L;
    energy(i) = Energy.calculate(L, W{i} * w);
end

fprintf('Initial energy: %i \nFinal energy: %i \nDifference: %i', energy(1), ener
```

Initial energy: 7.905530e+02
 Final energy: 7.905530e+02
 Difference: 5.764557e-06

Visualization

```
v = Visualization(tHandle, '6c/RK4');
v.plotX(t, W);
v.plotError(t, E);
v.plotEnergy(t, energy);
v.plotAccumulativeError(t, E);
```

6.2 Prosjektplan

Logistikk

Siden prosjektet tar opp 20% av et fag som går under 10 studiepoeng, antar vi at det tilsvarer 2 stp med arbeid. Vi antar at dette gir oss en ukentlig arbeidsmengde på ca 10-12 timer i uken per hode, siden vi er en gruppe på 3. Vi vil underveis justere dette basert på fremgangen vår i prosjektet.

Gruppen, bestående av tre personer, er inneforstått med at det kommer til å bli en del mer arbeid kontra de med 5 på hver gruppe. Søndager og torsdager er satt av som primære arbeidsdager. Utover uken blir det opp til hver enkelt å jobbe seg gradvis gjennom stoffet, og legge fram ulike forslag til løsninger på disse dagene. Kommunikasjon vil foregå over messenger, facebook og møter med faglærer på Microsoft Teams.

Litt om hver oppgavene

Oppgave 1

Går ut på å implementere en eksponentialfunksjon og en funksjon som regner ut energien til et legeme med et gitt trehetsmoment og en rotasjonsvektor. Deretter skal vi teste implementasjonen våre og regne ut trehetsmomentet til en T-nøkkel.

Dette legger grunnlaget for oppgavene som skal løses videre gjennom prosjektet. Ved å validere funksjonene som regner ut trehetsmomentet og rotasjons vektoren kan vi med sikkerhet vite at en eventuell feil vil komme fra et annet sted samtidig som det vil kunne gi oss en indikator på et virkelig scenario.

Oppgave 2

Her skal vi anta at vi har et kulelegeme, og utføre en eksakt løsning ved hjelp av formler vi har blitt gitt.

Oppgave 3

I og med at vi nettopp har løst likningene eksakt skal vi bruke en numerisk metode, Eulers formel, for å kunne se nøyaktigheten av denne samtidig gi en indikator på om vi har gjort oppgaven riktig.

Oppgave 4 & 5

I denne oppgaven skal vi se på både Runge Kutta-og Runge Kutta Fehlberg-metoden for numerisk løsning av systemet. Ved disse implementasjonen kan vi sammenligne med Eulers metode på systemet gitt i forrige oppgave for å feilestimering av metodene.

Oppgave 6

Her skal vi grafisk fremstille resultatene vi har funnet av X, med hensyn på t. Til slutt skal vi tolke disse grafene og komme frem til en konklusjon. Vi kan alternativt lage en animasjon av T-nøkkelen som viser resultatene og tolke dette.

Konklusjon

Til å konkludere med skal vi se på ulike numeriske metoder som RK4-, RK45 og Eulers-metode og videre sammenligne disse opp mot en felles eksakt løsning. Det vil bli brukt MatLab for eksakte verdier, og det vil da også være naturlig å bringe inn flyttalls problematikk. I rapporten vil det bli lagt fram grafiske framstillinger og animasjoner for å peke på ulike sammenhenger og komme til en konklusjon på hvorfor T-nøkkelen oppfører seg som den gjør når den roterer uten noen ytre påvirkning.

Erklæring

7.1 Håkon Harnes

Jeg har deltatt på gruppeprosjekt i TDAT3024 med Vetle Harnes og Jørgen Hollum. Dette er to metstudenter jeg har jobbet mye sammen med. Dette har ført til et godt samarbeid, og et godt resultat, til tross for at vi kun er tre på gruppen. Alle har vært aktive og bidratt gjennom løsning av oppgaver og rapportskriving.

Trondheim, 30. oktober 2020

Vetle Harnes

Vetle Harnes

Jørgen Hollum

Jørgen Hollum

Håkon Harnes

Håkon Harnes

7.2 Vetle Harnes

Jeg har deltatt på gruppeprosjekt i TDAT3024 med Håkon Harnes og Jørgen Hollum. Dette er to gode kompiser og studiesamarbeidspartnere som jeg har jobbet mye sammen med før. Vi valgte sammen å danne en gruppe på 3, selv om dette var færre en anbefalt, fordi vi kjenner hverandre og samarbeider godt. Vi har alle tatt en aktiv rolle, og samarbeidet godt gjennom prosjektløpet.

Trondheim, 30. oktober 2020

Håkon Harnes

Håkon Harnes

Jørgen Hollum

Jørgen Hollum

Vetle Harnes

Vetle Harnes

7.3 Jørgen Hollum

Jeg har deltatt på gruppeprosjekt i TDAT3024 med Vetle Harnes og Håkon Harnes. Dette er to kunnskapsrike kompiser jeg har jobbet sammen med i hele dataingeniør løpet. I starten av prosjekt fikk vi som studenter noenlunde velge hvor mange vi ville være på gruppe sammen med. Sammen sto vi for å være tre stykker i stedet for fem i og med at vi alle vet hvor hverandre er både jobbmessig og kunnskapsmessig. Gjennom prosjektet har alle deltatt og vært aktive på både rapportskriving og løsning av oppgaver. Inndeling av oppgaver ble gjort på bakgrunn av kunnskap og alle har tydelig tatt del i utviklingen. Jeg mener vi har minst gjort en like bra jobb som fremmer gruppene, da mye tid er lagt ned i prosjektet og alle oppgavers løsninger er dokumentert godt i rapporten. Vi har vært et godt team, hvor alle har tatt del i prosjektet.

Trondheim, 30. oktober 2020

Håkon Harnes

Håkon Harnes

Vetle Harnes

Vetle Harnes

Jørgen Hollum

Jørgen Hollum