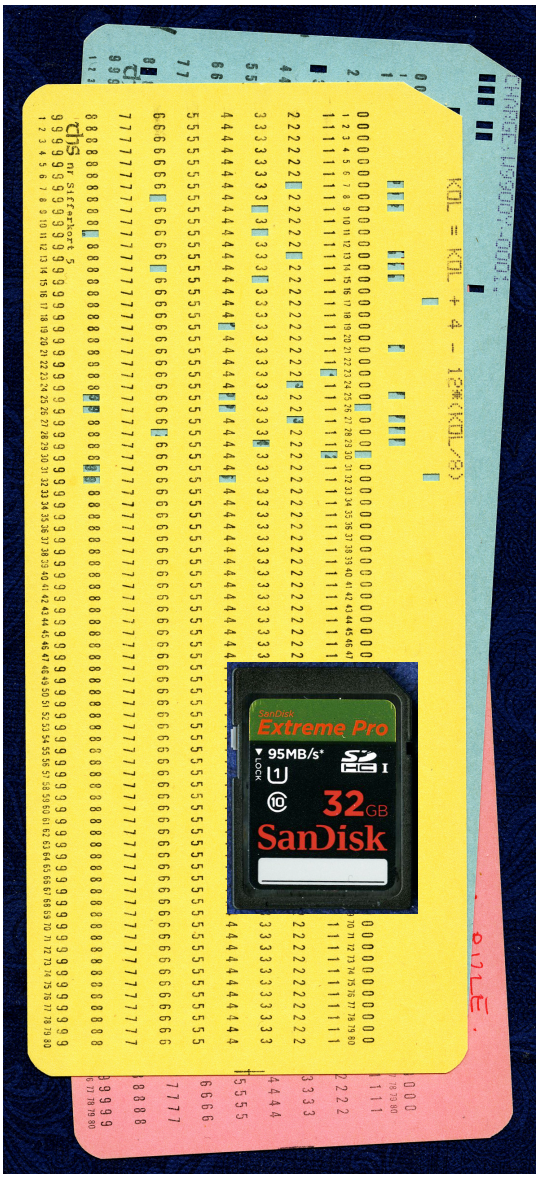
Kapittel 3

Numeriske løsningsmetoder



Eksempler på datakort (70% av naturlig størrelse), sammen med en moderne brikke (i naturlig størrelse) med lagerkapasitet svarende til 400 millioner hullkort (som hadde veid 950 tonn!).

I min studietid (1969-1974) hadde Norges største data-maskin 250 kb hukommelsete og fylte et helt rom. Vi laget programmer ved å punche, hver linje - ett hullkort. Kortbunken ble båret forsiktig til Abels hus (det var en katastrofe å miste den!). Noen timer opp til et døgn senere kunne vi hente resultatet i form av en utskrift på sideperforerte ark. En punsjefeil førte til at et kort måtte punches på nytt, og riktig kort i kortbunken byttes. Så var det en ny innlevering og ny venting. Gjett om uttesting av programmer tok lang tid!

I dag er situasjonen totalt forskjellig. Datamaskinen er allemanns eie. Programutvikling er utrolig effektiv sammenlignet med tidligere tider. Og numeriske metoder har blitt et like naturlig verktøy som analytisk matematikk.

Men verktøy er som verktøy flest: Det trengs opplæring i hvordan de brukes. I dette kapitlet skal vi først og fremst se hvordan svingeligningen og senere bølgelingningen kan løses på en god måte. Det er ikke nok å lese seg til hvordan ting kan gjøres. Praktisk trening må til for å få den nødvendige ferdigheten og rutinen.

[1]Deler av teksten er skrevet av David Skålid Amundsen som en sommerjobb for CSE 2008. Amundsens tekst er siden bearbeidet og utvidet av Arnt Inge Vistnes. Versjon 30012014.

3.1 Innledning

Når vi i "gamle dager" (det vil si for mer enn 15 år siden) skulle beregne bevegelsen til en matematisk eller fysisk pendel i et laveregrads fysikkurs, måtte vi nøye oss med "små utslag". Vi hadde den gang bare analytisk matematikk i verktøykassen, og da var det bare små utslag der bevegelsen er tilnærmet en ren harmonisk svingning vi kunne håndtere. Større utslag er det langt vanskeligere å håndtere rent analytisk, og dersom vi eventuelt legger inn kompliserende friksjon, finnes det rett og slett ikke noe analytisk løsning på problemet.

Når vi har lært oss å bruke numeriske løsningsmetoder, er det ofte omtrent like enkelt å bruke en realistisk, "ikke tilnærmet" beskrivelse av en bevegelse som en idealisert forenklet beskrivelse.

Denne boka er basert på at leseren allerede kjenner litt til hvordan vi løser f.eks. differensialligninger ved hjelp av numeriske metoder. Likevel tar vi med en rask repetisjon av noen av de enkleste løsningsmetodene slik at de som ikke har tidligere erfaring med numeriske løsningsmetoder, tross alt skal kunne henge med. Etter den raske gjennomgangen av de enkle metodene, bruker vi litt mer tid på en mer robust metode. I tillegg sier vi litt om hvordan disse metodene kan generaliseres til å løse partielle differensialligninger.

Det bør allerede her nevnes at de enkleste numeriske løsningsmetodene ofte er gode nok for f.eks. å beregne en kastbevegelse, også når friksjon er til stede. De enkleste metodene summerer imidlertid ofte opp feil og gir ganske dårlige resultat for svingebevegelser. Det er med andre ord ofte nødvendig å bruke litt avanserte numeriske metoder når vi gjør beregninger på svingninger og bølger.

Dette kapitlet er bygd opp på følgende måte:

Først gis en rask gjennomgang av de enkleste numeriske metodene som brukes ved løsning av differensialligninger. Dernest beskrives Runge-Kuttas metode av fjerde orden. Denne første delen av kapitlet er temmelig matematisk.

Deretter gis det et praktisk eksempel, og til slutt tar vi med eksempler på programkoder som kan anvendes i oppgaveløsning videre i boka.

3.2 Grunnleggende idé bak numeriske metoder

I mange deler av fysikken møter vi annen ordens ordinære differensialligninger:

$$\ddot{x} \equiv \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = f(x(t), \dot{x}(t), t). \tag{3.1}$$

I mekaniske system har differensialligningen ofte opphav i Newtons 2. lov. I elektriske svingekretser er det gjerne Kirchhoffs lov sammen med generalisert Ohms lov og komplekse

impedanser som er opphav til differensialligningene.

Uttrykket $f(x(t), \dot{x}(t), t)$ sier at f (i tilfelle x er en posisjonsvariabel og t tiden) er en funksjon av såvel tid, posisjon og hastighet.

Når vi løser annen ordens differensialligninger numerisk, betrakter vi ofte ligningen som en kombinasjon av to koblede første ordens differensialligninger:

$$\frac{\mathrm{d}\dot{x}}{\mathrm{d}t} = f(x(t), \dot{x}(t), t)$$

$$\frac{\mathrm{d}x}{\mathrm{d}t} = g(x(t), t)$$

hvor g er en funksjon som angir den tidsderiverte av x. Vi vil straks se noen enkle eksempler på dette i praksis.

3.3 Eulers metode og varianter av denne

Vi kan løse en differensialligning ved å angi en startverdi for den løsningen vi er interessert i, og bruke vår kunnskap om funksjonens deriverte for å beregne løsningen en kort tid Δt etterpå. Vi lar så den nye verdien fungere som ny startverdi for å beregne verdien som følger Δt etter dette igjen. Slik fortsetter vi helt til vi har beskrevet løsningen i så mange punkter n som vi er interessert i.

Utfordringen er å finne hvordan vi kan bestemme neste verdi ut fra den vi allerede kjenner. Det kan gjøres på en mer eller mindre raffinert metode. Den enkleste metoden er kanskje Eulers metode. Den tar utgangspunkt i den velkjente definisjonen av den deriverte:

$$\dot{x}(t) = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t}$$

Dersom Δt er tilstrekkelig liten, kan vi omskrive dette uttrykket og få:

$$x(t + \Delta t) \approx x(t) + \Delta t \dot{x}(t)$$

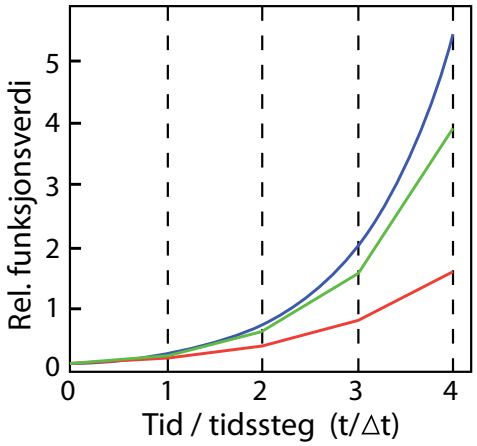
Anta at startverdiene er gitt ved: $(x_n, \dot{x}_n, t_n)$. Da følger for en første ordens differensialligning:

$$x_{n+1} = x_n + \dot{x}_n \Delta t$$

Ved å bruke en slik oppdateringsligning på både $x(t)$ og $\dot{x}(t)$, får vi den velkjente Eulers metode (i vår sammenheng for løsning av annen ordens differensiallikning):

$$\dot{x}_{n+1} = \dot{x}_n + \ddot{x}_n \Delta t$$
$$x_{n+1} = x_n + \dot{x}_n \Delta t$$

Vi har altså to koblede differensialligninger.



Figur 3.1: *Eulers enkle metode for å beregne en funksjon numerisk. Den øverste (blå) kurven er den korrekte. Den nederste (røde) kurven er beregnet ved hjelp av Eulers enkle metode, mens den midtre er beregnet med midtpunktsmetoden. Tidssteget er det samme i begge tilfeller og er valgt meget stort for å få fram prinsippet i metodene.*

Figur 3.1 skisserer hvordan metoden virker. Dette er den mest vanlige måten å lage en slik illustrasjon på, men den gir etter min mening bare en overfladisk forståelse. For hva skjer når vi får større og større avvik mellom den korrekte løsningen og den beregnede løsningen? Her er det noen detaljer vi bør kjenne til.

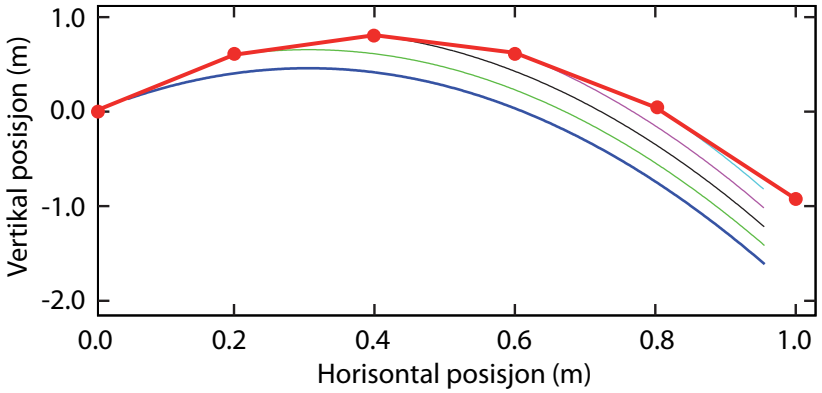
I figur 3.2 er det vist en figur lignende den i figur 3.1, men med en rekke nye kurver. Den mellomtykke blå kurven (nederst) viser hvordan et skrått kast vil forløpe for en initiell hastighet på 1.0 m/s i horisontal retning og 3.0 m/s i vertikal retning oppover. Beregningen er basert på en analytisk løsning av dette enkle problemet.

I figuren er det også tegnet inn løsningen når Eulers metode er benyttet med meget store tidssteg (0.2 s). Allerede etter ett tidssteg er den beregnede nye posisjonen temmelig langt fra det den skulle vært.

Etter dette tidssteget er det beregnet nye verdier for posisjon og hastighet i både horisontal og vertial retning. Det er disse verdiene som nå plugges inn i differensialligningen. Hadde vi beregnet banen for akkurat disse verdiene, ville vi fått løsningen gitt ved en grønn kurve (nest nederst). *Dette er en annen løsning av differensialligningen enn den vi opprinnelig startet ut med!*

Heller ikke nå klarer vi å følge denne nye løsningen på en god måte siden tidssteget er så stort, og når vi bruker Eulers metode enda en gang, får vi en posisjon (og hastigheter) som ligger temmelig langt også fra den andre løsningen av differensialligningen vi var innom.

Slik fortsetter det hele. For hvert nytt tidssteg er vi innom en ny løsning av differensialligningen, og i vårt tilfelle er feilen som gjøres systematisk og gir større og større feil etter hvert tidssteg.



Figur 3.2: *Beregning av et skrått kast ved hjelp av Eulers metode med stort tidssteg. Hvert nytt punkt som beregnes er utgangspunkt for en ny løsning av den opprinnelige differensialligningen. Se teksten for detaljer.*

Det kan vises at dersom vi reduserer tidssteget betydelig (!) i forhold til hva som er brukt i figur 3.2, blir løsningen langt bedre enn i figuren. Likevel er det ikke alltid nok å bare redusere tidssteget.

For det første kan vi ikke redusere tidssteget så mye at vi får problemer med hvor nøye tall kan angis på en datamaskin (uten at vi må anvende ekstremt tidkrevende teknikker). Når vi beregner $x_{n+1} = x_n + \dot{x}_n \Delta t$, må ikke bidraget $\dot{x}_n \Delta t$ bestandig være så lite at det bare kan bidra i det minst signifikante sifferet til x_{n+1}.

En annen begrensning ligger i den numeriske metoden i seg selv. Gjør vi systematiske feil som adderer seg til å bli værre og værre for hvert tidssteg, uansett hvor små tidsstegene er, får vi også problemer. Da må vi bruke andre numeriske metoder enn denne aller enkleste Euler.

En forbedret versjon av Eulers metode blir kalt *Euler-Cromers metode*. Anta at startverdiene er $(x_n, \dot{x}_n, t_n)$. Første trinn er identisk med Eulers enkle metode:

$$\dot{x}_{n+1} = \dot{x}_n + \ddot{x}_n \Delta t.$$

Andre trinn i Euler-Cromers metode adskiller seg imidlertid fra den enkle Euler: For å finne x_{n+1}, bruker vi $\dot{x}_{n+1}$ og ikke $\dot{x}_n$ som vi gjør i Eulers metode. Det gir følgende oppdateringslikning for x:

$$x_{n+1} = x_n + \dot{x}_{n+1} \Delta t.$$

Årsaken til at Euler-Cromers metode fungerer, og at den ofte (men ikke alltid) fungerer bedre enn Eulers metode er ikke triviell, og vi skal ikke gå nærmere inn på det her. Eulers metode fører ofte til at energien i det modellerte systemet ikke er bevart, men øker sakte men sikkert etterhvert. Dette problemet blir dramatisk redusert med Euler-Cromers metode, som gjør at den i de fleste tilfeller fungerer bedre.

En annen forbedring av Eulers metode, som er enda bedre enn Euler-Cromers metode, er *Eulers midtpunktsmetode*. Istedenfor å bruke stigningstallet *i begynnelsen* av intervallet, og bruke dette i hele intervallet, bruker vi stigningstallet *i midten* av intervallet. Ved å bruke stigningstallet i midten av intervallet vil vi normalt få et mer nøyaktig resultat enn ved å bruke stigningstallet i begynnelsen av intervallet når vi er ute etter å finne den gjennomsnittlige vekstraten.

I Eulers midtpunktsmetode bruker vi først stigningstallet i begynnelsen av intervallet, men istedenfor å bruke dette stigningstallet for hele intervallet, bruker vi det for halve intervallet. Så beregner vi stigningstallet i midten av intervallet og bruker dette for hele intervallet. Matematisk blir dette, ved å bruke samme notasjon som tidligere:

$$\begin{aligned} \dot{x}_{n+1/2} &= \dot{x}_n + f(x_n, \dot{x}_n, t_n)\Delta t/2, \\ x_{n+1/2} &= x_n + \dot{x}_n \Delta t/2. \end{aligned}$$

Her er altså $\dot{x}_{n+1/2}$ og $x_{n+1/2}$ verdien til den ukjente funksjonen og verdien til den deriverte av denne funksjonen midt i intervallet. Oppdateringsligningen for hele intervallet blir så:

$$\begin{aligned} \dot{x}_{n+1} &= \dot{x}_n + f(x_{n+1/2}, \dot{x}_{n+1/2}, t_{n+1/2})\Delta t, \\ x_{n+1} &= x_n + \dot{x}_{n+1/2}\Delta t. \end{aligned}$$

3.4 Runge-Kuttas metode

I Eulers metode fant vi neste verdi ved å bruke stigningstallet i starten av det steget vi skal ta. I Eulers midtpunktsmetode brukte vi stigningstallet i midten av steget vi skal ta. I begge tilfeller er det nokså lett å tenke seg at for noen funksjoner vil vi kunne få en systematisk feil som vil summeres opp til betydelige totale feil etter hvert som det gjennomføres mange etterfølgende beregninger. Det kan vises at feilen vi gjør blir vesentlig mindre dersom vi går over til å bruke enda mer raffinerte metoder for å finne neste verdi. En av de mest populære metodene kalles Runge-Kuttas metode av fjerde orden. Hele fire forskjellige estimater for stigningstallet, ett i begynnelsen, to i midten og ett i slutten blir da brukt for å beregne det gjennomsnittlige stigningstallet i intervallet. Dette gjør Runge-Kutta metoden mye bedre enn Eulers midtpunktsmetode, og siden den ikke er stort vanskeligere å programmere, er det denne som ofte blir brukt i praksis.

La oss se hvordan Runge-Kuttas metode av 4. orden fungerer, og hvordan den kan brukes for å løse en annen ordens differensialligning. (Helt til slutt i kapitlet følger pseudokode og full kode for et program som bruker Runge-Kuttas metode av fjerde orden.)

3.4.1 Beskrivelse av metoden

Runge-Kuttas metode er faktisk ikke så vanskelig å forstå, men du må antakelig lese detaljene som inngår to ganger for å se det. Vi skal først gi en matematisk gjennomgang, og så forsøke å oppsummere metoden ved hjelp av en figur (figur 3.3). La oss si litt om den matematiske notasjonen først. Ta utgangspunkt i differensialligningen gitt ved

$$\ddot{x}(t) = f(x(t), \dot{x}(t), t). \tag{3.2}$$

Eksempelvis, for en dempet fjærpendel fra kapittel 1 (hvor t ikke inngår eksplisitt fordi de fysiske lovmessighetene ikke endrer seg i tid), vil denne ligningen se slik ut:

$$\ddot{z}(t) = -\frac{b}{m}\dot{z}(t) - \frac{k}{m}z(t) \tag{3.3}$$

Anta at vi befinner oss i punktet $(x_n, \dot{x}_n, t_n)$ og at skrittlengden for tidsøkningen er Δt. I det videre kommer vi til å finne estimater for x_n, $\dot{x}_n$ og $\ddot{x}_n$, og hvilket nummer estimatet har i rekken blir angitt med en ny indeks. Det første estimatet til $\ddot{x}_n$ blir angitt som $a_{1,n}$, det første til $\dot{x}_n$ blir angitt som $v_{1,n}$ og det første til x_n blir angitt som $x_{1,n}$. Det andre estimatet til $\ddot{x}_n$ blir da angitt som $a_{2,n}$, og tilsvarende for resten av estimatene.

Vi kan finne første estimatet til $\ddot{x}_n$ ved å bruke ligning (3.2):

$$a_{1,n} = f(x_n, \dot{x}_n, t_n).$$

Samtidig er den førstederiverte kjent i starten av intervallet:

$$v_{1,n} = \dot{x}_n.$$

Neste skritt på veien er å bruke Eulers metode for å finne $\dot{x}(t)$ og $x(t)$ i midten av intervallet:

$$x_{2,n} = x_{1,n} + v_{1,n}\frac{\Delta t}{2}.$$

$$v_{2,n} = v_{1,n} + a_{1,n}\frac{\Delta t}{2},$$

Videre kan vi finne et estimat for den andrederiverte i midten av intervallet ved å bruke $v_{2,n}$, $x_{2,n}$ og ligning (3.2):

$$a_{2,n} = f(x_{2,n}, v_{2,n}, t_n + \Delta t/2).$$

Neste skritt på veien i Runge-Kuttas metode er å bruke den nye verdien for den andrederiverte i midten av intervallet til å finne et nytt estimat for $x(t)$ og $\dot{x}(t)$ i midten av intervallet ved hjelp av Eulers metode:

$$x_{3,n} = x_{1,n} + v_{2,n}\frac{\Delta t}{2},$$

$$v_{3,n} = v_{1,n} + a_{2,n}\frac{\Delta t}{2}.$$

Med det nye estimatet for $x(t)$ og $\dot{x}(t)$ i midten av intervallet kan vi finne et nytt estimat for den andrederiverte i midten av intervallet:

$$a_{3,n} = f(x_{3,n}, v_{3,n}, t_n + \Delta t/2).$$

Ved hjelp av det nye estimatet for den andrederiverte i tillegg til estimatet for den førstederiverte i midten av intervallet kan vi nå bruke Eulers metode for å finne et estimat for $x(t)$ og $\dot{x}(t)$ i slutten av intervallet. Det gjøres da slik:

$$x_{4,n} = x_{1,n} + v_{3,n}\Delta t,$$

$$v_{4,n} = v_{1,n} + a_{3,n}\Delta t.$$

Til slutt kan vi på tilsvarende måte som før finne et estimat for $\ddot{x}(t)$ i slutten av intervallet ved hjelp av disse nye verdiene:

$$a_{4,n} = f(x_{4,n}, v_{4,n}, t_n + \Delta t)$$

Vi kan nå regne ut et vektet gjennomsnitt av estimatene, og vi får da et rimelig godt estimat for den gjennomsnittlige verdien til den andrederiverte og førstederiverte i intervallet:

$$a_n = \frac{1}{6}\left(a_{1,n} + 2a_{2,n} + 2a_{3,n} + a_{4,n}\right) \tag{3.4}$$

$$v_n = \frac{1}{6}\left(v_{1,n} + 2v_{2,n} + 2v_{3,n} + v_{4,n}\right) \tag{3.5}$$

Ved hjelp av disse gjennomsnittene, som er ganske gode tilnærminger til middelverdiene av stigningstall over hele intervallet, kan vi bruke Eulers metode for å finne et godt estimat for $x(t)$ og $\dot{x}(t)$ i slutten av intervallet:

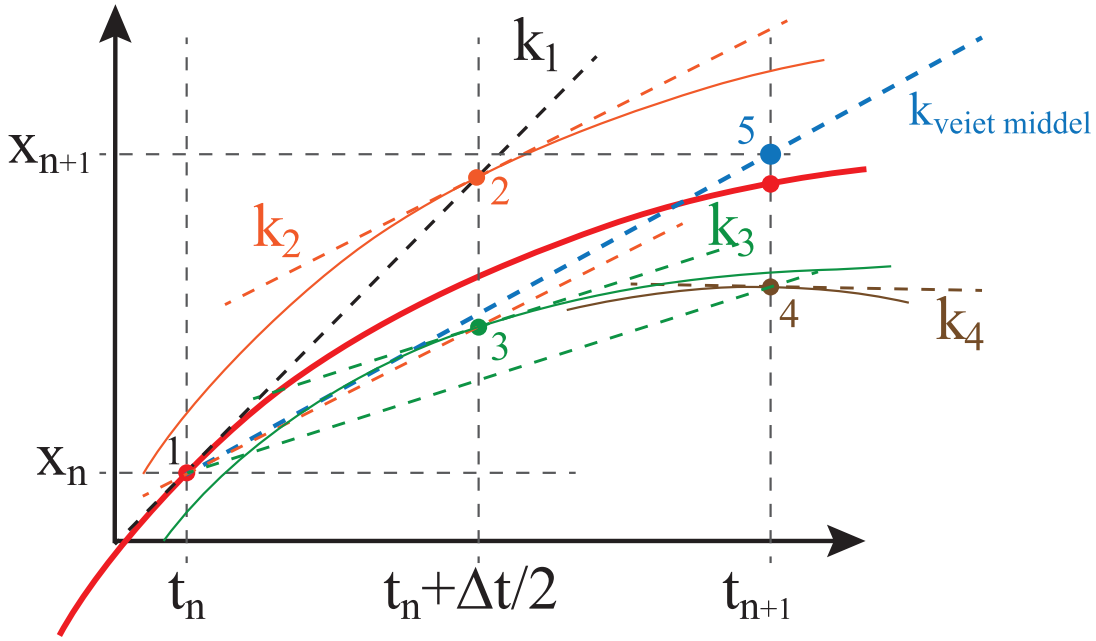
$$x_{n+1} = x_n + v_n\Delta t \tag{3.6}$$

$$v_{n+1} = v_n + a_n\Delta t \tag{3.7}$$

$$t_{n+1} = t_n + \Delta t \tag{3.8}$$

Dette er da ekvivalent med startverdiene for neste intervall.

I Runge-Kuttas metode (se figur 3.3) henter vi ut mye mer informasjon fra differensialligningen enn i Eulers metode. Dette gjør Runge-Kuttas metode betydelig mer stabil enn Eulers metode, Euler-Cromers metode og Eulers midtpunktsmetode. Runge-Kuttas metode krever ikke så enormt mye mer ressurser for en datamaskin å utføre, samtidig er den forholdvis enkel å programmere. Runge-Kuttas metode, i en eller annen variant, er derfor ofte den metoden vi først tyr til når vi ønsker å løse ordinære differensialligninger numerisk. Programmeringen av den grunnleggende delen av Runge-Kutta gjøres nærmest en gang for alle. Det er vanligvis bare en liten fil som endres fra ett problem til et annet. Filen angir nøyaktig differensialligningene som skal benyttes i akkurat de beregningene som da skal gjennomføres. Se eksempelkode senere i kapitlet.



Figur 3.3: *Oppsummering av fjerde ordens RungeKutta's metode. Se tekst.*

Det kreves konsentrasjon for å skjønne figuren fullt ut: I punkt 1 (x_n, t_n) beregnes stigningstallet k_1. Vi følger tangenten i punktet 1 et halvt tidssteg, og kommer til punkt 2 (rosa). Dette punktet ligger på en annen løsning av differensialligningen (tynn rosa linje) enn den vi søker. Vi beregner stigningstallet k_2 i punkt 2 for denne løsningen (rosa stiplet linje). Vi trekker så en linje fra punkt 1 igjen, men nå med stigningstallet vi fant i punkt 2. Igjen går vi bare halve tidssteget, og finner punkt 3 (grønt). Det er enda en ny løsning av differensialligningen som går gjennom dette punktet (tynn grønn linje). Vi beregner stigningstallet k_3 i punkt 3 for denne løsningen (stiplet grønn linje). Vi trekker så en linje gjennom punkt 1 igjen, men nå med stigningstallet vi nettopp fant. Nå går vi hele tidssteget og ender i punkt 4 (brunt). Igjen er det en ny løsning av differensialligningen som går gjennom dette punktet. Vi beregner stigningstallet k_4 for denne løsningen i punkt 4.

Vi har nå alt i alt beregnet fire ulike stigningstall, ett ved t_n, ett ved t_{n+1}, og to midt mellom. Vi beregner så en veiet middelverdi med samme vekting som i ligning (3.4) og (3.5), og bruker denne verdien som stigningstall fra punkt 1 (x_n, t_n) et helt tidsintervall og kommer til punkt 5 (blått) (x_{n+1}, t_{n+1}) . Dette punktet ligger vanligvis ikke nøyaktig på den løsningen av differensialligningen vi søker, men likevel mye nærmere enn noen av de andre tre løsningene vi var innom underveis. Med utgangspunkt i punkt (x_{n+1}, t_{n+1}) bruker vi samme omfattende prosedyre enda en gang for å finne neste punkt (x_{n+2}, t_{n+2}) på den tilnærmede korrekte løsningen vi beregner numerisk steg for steg.

For mer stoff om Runge-Kuttas metode se Tom Lindstrøms "Kalkulus" eller Wikipedia.

3.5 Partielle differensialligninger

Mange fysiske problemer er beskrevet ved hjelp av partielle differensialligninger, kanskje de mest kjente er Maxwells ligninger, Schrödingerligningen og bølgeligningen. Med partielle differensialligninger menes at den ukjente funksjonen består av flere variable og at derivasjon med hensyn på disse inngår i differensialligningen.

Det finnes flere forskjellige løsningsmetoder for partielle differensialligninger, men den enkleste og letteste å forstå er nok den såkalte "endelig-differanse-metoden" (finite difference method). Den går ut på å erstatte differensialene i differensialligningen med endelige differanser. Betrakt den enkle differensialligningen

$$\frac{\partial y}{\partial x} = K\frac{\partial y}{\partial t}. \tag{3.9}$$

Den enkleste måten å gjøre om differensialene i denne ligningen til diskrete differensialer, er å bruke definisjonen av den deriverte, som vi har gjort tidligere. Ligningen over blir da

$$\frac{y(x+\Delta x,t)-y(x,t)}{\Delta x} = K\frac{y(x,t+\Delta t)-y(x,t)}{\Delta t}.$$

Denne ligningen kan så løses med hensyn på $y(x,t+\Delta t)$, som gir

$$y(x,t+\Delta t) = y(x,t) + \frac{\Delta t}{K\Delta x}\left(y(x+\Delta x,t)-y(x,t)\right).$$

Dersom $y(x,t)$ er kjent ved et tidspunkt $t=t_0$, slik at $y(x,t_0)=f(x)$, gir ligningen over funksjonsverdien ved neste tidspunkt $y(x,t_0+\Delta t)$. Men, legg merke til at vi også bruker funksjonsverdien ved en annen x enn den vi gjør beregning på i uttrykket over. Det betyr at vi får et problem når vi skal beregne funksjonsverdier for x nær yttergrensen for det området vi gjør beregninger for. Av ligningen over ser vi at vi må vite hva funksjonen var ved neste x-koordinat ved forrige tidspunkt, og i det ytterste x-punktet går ikke dette.

Dette medfører at vi må kjenne *randbetingelsene*[a], det vil si tilstanden til systemet i grensen av området vi ser på, for at problemet skal ha en entydig løsning. Disse må avklares før beregningene i det hele tatt kan begynne. Randbetingelser finnes også for ordinære differensialligninger, men i partielle differensialligninger er de mye mer fremtredende og kommer i tillegg til initialbetingelsene.

[a] Initialbetingelser og randbetingelser er to forskjellige ting, og må ikke blandes sammen. Initialbetingelser sier noe om hvilken tilstand systemet befinner seg i helt i begynnelsen av beregningene, og må også brukes her. Randbetingelsene sier noe om systemets tilstand ved endepunktene av beregningene ved *alle* tidspunkt.

De endelige differensialene vi brukte over blir derimot sjeldent brukt, da de kan erstattes av noe som er bedre og ikke stort mer vanskelig å forstå. Istedenfor å bruke Eulers metode i differensialene som over, brukes Eulers midtpunktsmetode, som reduserer feilen i

beregningene betraktelig. Dersom vi gjør dette, blir diskretiseringen av ligning (3.9) slik:

$$\frac{y(x+\Delta x,t)-y(x-\Delta x,t)}{2\Delta x}=K\frac{y(x,t+\Delta t)-y(x,t-\Delta t)}{2\Delta t}.$$

Det er ikke så vanskelig å forstå at resultatet nå vil bli bedre, for istedenfor beregne den gjennomsnittlige veksten gjennom det aktuelle punktet og neste punkt, brukes den gjennomsnittlige veksten gjennom forrige og neste punkt. På samme måte som isted kan denne ligningen løses med hensyn på $y(x,t+\Delta t)$, og resultatet blir:

$$y(x,t+\Delta t)=y(x,t-\Delta t)+\frac{\Delta t}{K\Delta x}\left[y(x+\Delta x,t)-y(x-\Delta x,t)\right]$$

Vi ser at vi får samme problem med randbetingelser som over, faktisk krever den en randbetingelse til, også i begynnelsen av x-gridet. Siden dette er et problem som omhandler en romlig dimensjon, må vi angi to randbetingelser for at løsningen skal være entydig (det er to randpunkter). For å bruke den første oppdateringsligningen må vi altså ta hensyn til den ande randbetingelsen også.

På samme måte som vi erstattet førstederiverte med et endelig differensial, kan n-tederiverte tilnærmes på samme måte. Et eksempel er den andrederiverte som kan tilnærmes med følgende differensial:

$$f''(x)\approx\frac{f(x+\Delta x)-2f(x)+f(x-\Delta x)}{\Delta x^2}$$

Bevis: Taylorutviklingen til funksjonen $f(x)$ om $x=a$ blir:

$$f(x)=f(a)+f'(a)(x-a)+\frac{f''(a)}{2}(x-a)^2+\ldots$$

En tilnærming til $f(a+\Delta x)$ er gitt ved

$$f(a+\Delta x)\approx f(a)+f'(a)(a+\Delta x-a)+\frac{f''(a)}{2}(a+\Delta x-a)^2,$$

som gir

$$f''(a)\approx\frac{2}{\Delta x^2}\left(f(a+\Delta x)-f(a)-f'(a)\Delta x\right).$$

Ved å sette inn for $f'(a)$ ved å bruke ligningen over for den førstederiverte, gir det

$$f''(a)\approx\frac{2}{\Delta x^2}\left(f(a+\Delta x)-f(a)\right)-\frac{2}{\Delta x^2}\left(\frac{f(a+\Delta x)-f(a-\Delta x)}{2\Delta x}\Delta x\right).$$

Ved å rydde får vi formelen

$$f''(a)\approx\frac{f(a+\Delta x)-2f(a)+f(a-\Delta x)}{\Delta x^2}. \tag{3.10}$$

Vi kan også bevise samme uttrykk ved å ta utgangspunkt i den deriverte av den deriverte, og anvende den enkleste definisjonen av den deriverte. Metoden har den fordelen at det fremgår tydeligere hvorfor vi må kjenne funksjonsverdien i (minst) tre punkt for at vi overhodet skal kunne beregne en annenderivert.

Som med de ordinære differensialligningene kan vi gå videre og bruke metoder som gir et enda bedre resultat.

Det finnes en rekke metoder tilgjengelig for ulike deler av fysikken. Interesserte henvises til spesielle kurs/bøker i numeriske beregninger.

3.6 "Dimensjonsløs" differensialligning *

Når vi løser fysiske problemer numerisk, brukes ofte såkalte "dimensjonsløse variable". Grunnen er at vi lett kan miste numerisk presisjon om vi ikke gjør dette. Et reelt tall i en datamaskin kan bare være innenfor et bestemt intervall, og dersom vi i beregningene kommer nær grensene for hva som kan representeres med full presisjon, går det ut over nøyaktigheten i beregningene. Selv om et tall i seg selv kan være langt fra grensene, vil vi kunne støte på høyere potenser av tallet i en eller annen mellomregning, og da kan skaden være gjort.

Av den grunn forsøker vi å holde alle variable på et nivå som ikke er alt for langt fra 1. Når vi f.eks. regner på en atomkjerne og har dimensjoner langt ned i 10^{-15} m, eller gjør beregninger innen kosmologi og har avstander på over en milliard lysår, dvs over $9.46 \cdot 10^{24}$ m, er bruk av såkalte dimensjonsløse variable ofte gunstig.

En annen grunn til å bruke dimensjonsløse variable, er at vi kan løse et problem for en hel klasse problemer i en og samme beregning, og at vi kan finne løsning av alle varianter av et problem ved bare å skalere løsningen som er oppnådd ved hjelp av dimensjonsløse variable.

I vår sammenheng er det ikke mye å vinne på å bruke dimensjonsløse variable. Grunnen er at vi oftest arbeider med fenomener fra dagligdagse fenomener der SI-enhetene meter og sekund er meget velegnet. Vi skal heller ikke studere mange varianter av eksakt samme type problem, og får derfor ikke noe særlig rasjonaliseringsgevinst ved å velge dimensjonsløse variable. Likevel nevner vi hovedprinsippet som benyttes slik at du kan velge selv om du vil bruke teknikken eller ikke.

Som et eksempel velger vi å ta utgangspunkt i differensialligningen som beskriver en dempet svingning av en fjærpendel (ligning (1.8) i kapittel 1). Den kan skrives som:

$$\frac{d^2z}{dt^2} + 2\gamma\frac{dz}{dt} + \omega^2 z = 0 \tag{3.11}$$

Her er $2\gamma = b/m$, det vil si friksjonskoeffisienten dividert på massen, mens $\omega^2 = k/m$ hvor k er fjærstivheten.

Alle ledd i summen skal ha samme enhet, nemlig $\mathrm{m/s^2}$. Du bør verifisere dette selv.

Vi vil nå skalere alle lengde- og tidsangivelser, og skriver:

$$z = z_0\hat{z}$$

$$t = t_0\hat{t}$$

Her er z_0 en referanselengde, angitt med riktig benevning/enhet. Skal vi studere en fjærpendel med amplitude f.eks. 50 cm, kunne vi latt z_0 være 0.5 m. Men siden dette er en størrelse så nær opp til SI-enheten 1 m, og så lenge det ikke er andre faktorer i ligningen som ville bli forenklet av et annet valg, vil det i dette tilfellet være naturlig å la z_0 være lik 1 m. I så fall vil $\hat{z}$ være ubenevnt, nemlig lik måletallet for en lengde (posisjon rel. likevektspunktet) når måleenheten er 1 m.

For tiden er det litt annerledes. Her kan vi velge SI-enheten 1 sek, eller vi kan velge en tidsenhet som er typisk for dempingen (dvs som bestemmes ut fra gamma, dvs fra friksjonsleddet), eller vi kan velge en tidsenhet som er typisk for selve svingetiden for pendelen dersom det ikke var friksjon. Vi velger siste variant, og for å få enklest mulig uttrykk velger vi helt konkret å la

$$t_0 = \frac{1}{\omega}$$

Igjen vil $\hat{t}$ være et dimensjonsløst måletall for tidsangivelser gitt i tidsenhet lik $1/\omega$.

Da er det på tide å sette inn for disse uttrykkene i den opprinnelige differensialligningen, og vi starter med å finne et uttrykk for hastighet med de nye variablene.

$$\frac{dz}{dt} = \frac{d(z_0\hat{z})}{d\hat{t}}\frac{d\hat{t}}{dt}$$

z_0 er en konstant og kan settes utenfor. Det siste leddet finnes lett når vi ser at $\hat{t} = t/t_0$ hvor også t_0 er en konstant. Da følger:

$$\frac{dz}{dt} = \frac{z_0}{t_0}\frac{d\hat{z}}{d\hat{t}}$$

På lignende måte kan vi finne et uttrykk for akselerasjonen, og resultatet blir:

$$\frac{d^2z}{dt^2} = \frac{z_0}{t_0^2}\frac{d^2\hat{z}}{d\hat{t}^2}$$

Vi setter nå inn disse uttrykkene i den opprinnelige differensialligningen, og setter også inn at $t_0 = \frac{1}{\omega}$. Resultatet blir da:

$$z_0\omega^2\frac{d^2\hat{z}}{d\hat{t}^2} + 2\gamma z_0\omega\frac{d\hat{z}}{d\hat{t}} + z_0\omega^2\hat{z} = 0$$

Vi "forkorter" med faktoren $z_0\omega^2$ og får:

$$\frac{d^2\hat{z}}{d\hat{t}^2} + 2\frac{\gamma}{\omega}\frac{d\hat{z}}{d\hat{t}} + \hat{z} = 0 \tag{3.12}$$

Sammenligner vi nå den opprinnelige ligningen (3.11) med den dimensjonsløse ligningen (3.12), ser vi at vi har fått en viss forenkling. Ved å løse ligning (3.12), kan vi behandle alle dempede svingninger hvor γ/ω er identiske, ved å bare skalere lengder og tider i tråd med hva vi har gjort i utledningen. I dette tilfellet er det imidlertid praktisk talt ingen rasjonaliseringsgevinst å hente ut fra dette.

For å være konkret må vi bruke $\hat{t} = \omega t$ i beregningene, og når resultatet er fremkommet, må vi transformere tilbake til SI-enheter ved å bruke relasjonen $t = \hat{t}/\omega$. For posisjoner/lengder kan det i utgangspunktet se ut for at uansett hvilken z_0 vi hadde valgt, ville løsningen være identisk. *Men* dersom vi husker at vi må spesifisere en startposisjon (rel. likevektspunktet) for å starte beregningen, innser vi at valget av z_0 betyr noe for hvilke tall beregningene ender opp med likevel. Velges en z_0 forskjellig fra 1 m, må initialverdien skaleres tilsvarende, og alle posisjoner må tilbakeskaleres etterpå. Initialbetingelsene må nemlig også angis på dimensjonsløs form når vi bruker ligning (3.12). Det betyr også at dersom vi angir initialbetingelsene i form av posisjon z_1 og hastighet v_1 ved tiden $t = 0$, må initialbetinglesene for ligningen (3.12) henholdsvis være $\hat{z} = z_1/z_0$ og

$$\hat{v}_1 \equiv \left(\frac{d\hat{z}}{d\hat{t}}\right)_1 = \frac{t_0}{z_0}v_1$$

3.7 Eksempel på numerisk løsning: Enkel pendel

La oss ta et konkret eksempel, nemlig en tilnærmet matematisk pendel som kan svinge med vilkårlig stort utslag (opp til $\pm\pi$) uten å klappe sammen (dvs. snora er "stiv"). Vi regner at all masse ligger i en bitte liten kule i enden av snora.

Mekanikken sier oss at kraften som trekker pendelen langs banen mot likevektspunktet er

$$F_\theta = -mg\sin(\theta)$$

når vinkelutslaget er θ. Kraftmomentet omkring opphengspunktet for pendelen med lengde L er da:

$$\tau = -mgL\sin(\theta)$$

Spinnsatsen anvendt omkring opphengspunktet gir:

$$\tau = I\alpha = I\ddot{\theta}$$

Her er $\alpha = \ddot{\theta}$ vinkelakselerasjonen og I er treghetsmoment om aksen. Vi velger den enkleste beskrivelse:

$$I = mL^2$$

og ender opp med den endelige differensialligningen:

$$mL^2\ddot{\theta} = -mgL\sin(\theta)$$

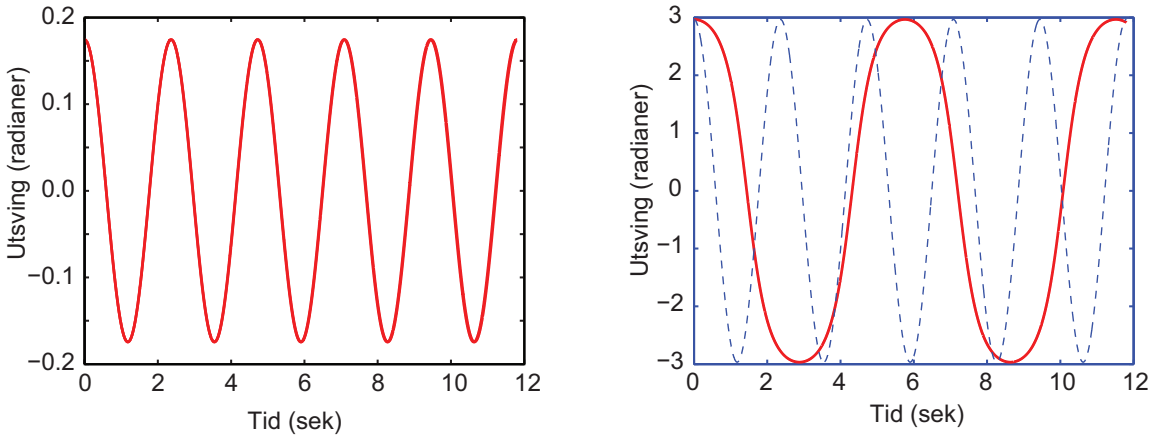
$$y\ddot{\theta} = -\frac{g}{L}\sin(\theta)$$

I mekanikken ble denne ligningen løst ved å anta at vinkelen θ er så liten at $\sin(\theta) \approx \theta$. Løsningen ble en enkel harmonisk bevegelse med svingefrekvens (vinkelfrekvens) gitt ved:

$$\omega = \sqrt{\frac{g}{L}}$$

Tilnærmingen $\sin(\theta) \approx \theta$ ble gjort for å kunne bruke analytiske metoder. Denne tilnærmingen var ikke helt nødvendig i akkurat dette spesielle tilfellet, fordi vi *kan* løse den opprinnelige differensialligningen analytisk også for store vinkler ved å benytte oss av rekkeutviklingen til sinusfunksjonen. Det er imidlertid enklere å bruke numeriske metoder.

Resultatet av numeriske beregninger hvor vi bruker fjerde ordens Runge-Kutta, er vist i figur 3.4. Vi ser at bevegelsen er nær harmonisk for små vinkelutslag, men svært forskjellig fra en sinus for et stort utslag. Dessuten har periodetiden endret seg mye. Merk at vi i høyre del av figuren har valgt en bevegelse der pendelen *nesten* når retningen "rett opp" både på "framovertur" og "bakovertur" (utsving nær $+\pi$ og $-\pi$).



Figur 3.4: *En pendel svinger harmonisk når utslaget er lite, men svingeforløpet endres mye når svingevinkelen øker. Også svingetiden endres. Forøvrig: Se teksten.*

Dersom vi ønsket å inkludere friksjon ved beskrivelsen av pendelbevegelsen, ville det representere et mer komplisert uttrykk for den effektive kraften enn vi hadde i vårt tilfelle. For ikke-lineær beskrivelse av friksjon finnes det ingen analytiske løsning.

Siden hovedstrukturen i en numerisk løsning ville være den samme, uansett hvilken beskrivelse vi har av effektiv kraft som virker på systemet, kan de mer kompliserte fysiske forholdene ofte håndteres forbausende lett med numeriske løsningsmetoder (se figur i en av oppgavene bak i kapitlet).

Det er en ekstra bonus ved numeriske løsningsmetoder: Kraften som virker blir mer sentral i vårt arbeid med å finne løsningen, og derved selve fysikken i problemet! Hvilken kraft gir hvilket resultat? Numerikken blir den samme, og vi behøver ikke pønske ut ulike til dels vrine analytiske løsningsmetoder og triks som er forskjellige for hvert uttrykk vi har for kraften. Fokus blir der den bør være: på utgangspunktet som er effektiv kraft og differensialligningen som gjelder, hvilke initialbetingelser vi har, og så hvilket resultat vi får.

3.8 Test av implementering

Det er fort gjort å gjøre en feil, enten i den analytiske matematikken, eller når vi lager dataprogrammet med numerisk løsningsmetoder. Vi har meget tragiske eksempler på hvor galt det kan gå i slike sammenhenger!

Det er derfor meget viktig å teste den numeriske implementeringen for å luke ut så mange feil vi bare kan. Det er ofte lettere sagt enn gjort! Vi bruker jo ofte numeriske metoder fordi vi ikke har noen analytiske metoder å falle tilbake på.

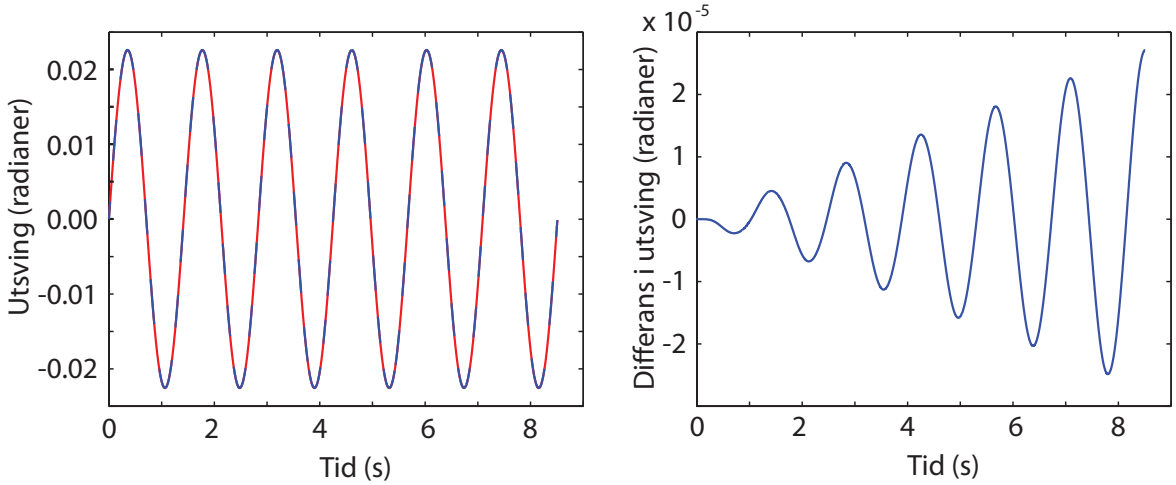
I vårt tilfelle med den mekaniske pendelen, er det likevel et triks vi kan gjøre. Det finnes en analytisk løsning som er tilnærmet riktig for *små* utslag. For *det* spesialtilfellet, kan vi teste om den numeriske løsningen blir omtrent den samme som den analytiske. Dersom det er uoverenskomst mellom disse to løsningene, er det opplagt en feil et eller annet sted.

Dersom den numeriske løsningen er lik den analytiske i dette spesialtilfellet, er det dessverre ikke et bevis på at programmet er feilfritt! Implementeringen av det spesielle som gjelder ut over spesialtilfellet, kan likevel være feil. Her er det nødvendig å vurdere de fysiske prediksjonene: virker de rimelige eller urimelige? Det er ofte umulig å være helt sikker på at et dataprogram er fullstendig korrekt. Innen informatikk er det spesielle teknikker som kan brukes i en del tilfeller. Vi kan ikke gå inn på disse. Hovedpoenget er at vi må være ydmyke og åpne for at feil kan finnes, og at vi forsøker å teste implementeringen av numeriske metoder hver gang vi utvikler et dataprogram.

Som et eksempel skal vi nå forsøke å sjekke programmet vi brukte i beregningene som førte til figur 3.4. Det er bare tilfellet der utslaget er lite vi kan bruke i testen vår.

I figur 3.5 er det til venstre vist resultat av de numeriske beregningene (rød kurve) sammen med analytisk løsning (stiplet blå kurve) i et tilfelle når pendelutsvinget er lite (maksimalt ± 0.023 radianer). Det er ikke mulig å se forskjeller mellom de to kurvene.

Å plotte analytisk og numerisk løsning i samme diagram er en vanlig måte å sjekke at



Figur 3.5: *Sammenligning mellom analytisk og numerisk løsning av en pendelbevegelse. For forklaringer: Se teksten.*

to løsninger er lik hverandre. Det er imidlertid en svært grov test, for det er begrenset oppløsning i en grafisk framstilling. I høyre del av figuren har vi valgt en bedre test. Her er *differansen* mellom analytisk og numerisk resultat angitt, og vi ser at det sannelig var litt forskjeller selv om vi ikke så dette i venstre del.

Vi kan nå se at forskjellen øker på en systematisk måte. Etter seks perioder har forskjellen økt til $2.7 \cdot 10^{-5}$ radianer. Er dette en indikasjon på at dataprogrammet vårt er feil?

Vi vet imidlertid at den analytiske løsningen egentlig bare var tilnærmet rett, og tilnærmingen bør bli bedre desto mindre utslaget er. Vi kan da redusere utslaget og se hva som skjer. Beregninger viser at dersom utslaget blir redusert til $1/10$ av det vi har i figuren, reduseres maksimal forskjell etter seks perioder til $1/1000$ av det vi hadde i stad. Reduserer vi utslaget til $1/100$ av det opprinnelige, reduseres maksimal forskjell til 10^{-6} av den opprinnelige forskjellen. Vi ser at numerisk og analytisk løsning blir mer og mer lik hverandre, og på en slik måte som vi måtte forvente. Dersom vi atpåtil tar en titt på rekkeutviklingen for sinus-funksjonen, gir det oss enda et holdepunkt på at resultatene våre er slik vi måtte forvente.

Vi kan da føle oss rimelig sikre på at programmet oppfører seg som det bør for små vinkler, og at det synes å håndtere økte vinkler som det bør, i alle fall så lenge de er små.

[♠ ⇒ Det er også en annen test vi ofte må gjøre i forbindelse med numeriske beregninger. Vi valgte å bruke 1000 tidssteg innen hver periode i beregningene som ligger bak figur 3.4 og 3.5. For beregninger som strekker seg over svært mange perioder, kan vi ikke bruke så små tidssteg. Går vi ned til f.eks. 100 beregninger per periode, vil resultatet vanligvis fortsatt være ok (avhengig av hvilke krav vi setter), men går vi f.eks. ned til 10 tidssteg per periode, vil resultatet nesten garantert avhenge mye av valget av tidssteg. Vi må ofte gjøre et sett beregninger for å forsikre oss om at "oppløsningen" i beregningene er akseptabel (verken for høy eller for lav).⇐ ♠]

3.9 Krav til reproduserbarhet

I dag er det lekende lett å endre på et program fra en kjøring til den neste. Det gir ekstra utfordringer som må tas alvorlig. Når vi gjør beregninger som skal brukes i en vitenskapelig artikkel, en masteroppgave, en prosjektoppgave, ja nærmest i hvilken som helst sammenheng hvor vårt program brukes, må vi kjenne eksakt program og parametre som er brukt dersom resultatene skal ha full verdi. I eksperimentell fysikk vet vi at det er viktig å angi i labjournalen alle detaljer om hvordan eksperimentene er gjennomført. Hensikten er at det skal være mulig å etterprøve resultatene vi får. Dette er helt essentielt for å få reproduserbarhet og for at vi skal kunne oppnå såkalt "intersubjektivitet" (at resultatet skal være uavhengig av hvilken person som faktisk gjennomførere eksperimentet), noe som er ekstremt viktig innen vitenskap og utvikling.

I eksperimentell virksomhet faller man iblant for fristelsen til å ikke notere alle relevante detaljer mens eksperimentet foregår. Vi er interessert i resultatet og tenker gjerne at når vi har kommet litt lenger og fått enda bedre resultater, *da* skal vi skrive ned alle detaljer. En slik praksis fører ofte med seg en del frustrasjon på et senere tidspunkt, for plutselig oppdager vi at noen viktige opplysninger faktisk aldri ble notert. I verste fall kan resultatet bli at vi må gjøre eksperimenter om igjen og lete oss fram til betingelser som var slik de var i et tidligere eksperiment hvor resultatene viste seg å være spesielt interessante.

Moderne bruk av numeriske metoder kan på flere måter sammenlignes med eksperimentelt arbeid i laboratoriet. Vi tester ut hvordan ulike parametre i beregningen innvirker på resultatene, og vi bruker ulike numeriske metoder på tilsvarende måte som vi bruker ulike måleinstrumenter og protokoller i eksperimenter. Det betyr at det stilles like strenge krav til dokumentasjon for den som driver med numeriske metoder som for eksperimentalisten.

For å etterkomme dette kravet bør vi innarbeide gode vaner i programmeringen. En måte vi kan etterkomme krav om reproduserbarhet, er å gjøre følgende:

- Angi i programkoden et "versjonsnummer" for programmet ditt.
- I resultatfilen du genererer må versjonsnummeret legges inn automatisk.
- Hver gang du endrer programmet forut for en beregning du vil ta vare på, må versjonsnummer oppdateres.
- Hver versjon av programmet (som faktisk brukes i praksis) må lagres på disk slik at det alltid er mulig å kjøre om igjen et program med gitt versjonsnummer.
- Parametre som brukes og som varierer fra kjøring til kjøring innenfor samme versjon av programmet, må skrives ut til en fil sammen med resultatet av kjøringen.

Lever vi opp til disse reglene, vil vi alltid kunne gå tilbake og reprodusere de resultatene vi fikk. Det er her antatt at resultatene er uavhengig av hvilken datamaskin beregningene foregår ved. Dersom vi har mistanke om at en kompilator eller bakenforliggende progam

eller operativsystem kan ha svakheter, kan det være aktuelt å angi også tilleggsopplysninger om dette sammen med resultatene (i en resultatfil).

I programekspemplene i denne boka er det de fleste steder *ikke* tatt med i koden de linjene som trengs for dokumentasjon av parametre og versjonsnummer. Grunnen er at programsnuttene som er angitt først og fremst er ment å vise hvordan selve beregningene kan gjennomføres. I eksempelprogrammet "Eksempelprogram i Matlab hvor Runge-Kutta brukes i praksis" er det imidlertid vist et eksempel på hvordan dokumentasjon om versjon og parametre kan håndteres.

3.10 Arbeidsgang ved numeriske metoder

Mange av problemstillingene vi møter i denne boka er knyttet til oppintegrering av differensialligninger som beskriver de prosessene vi er interessert i. Det er mange mulige måter å foreta denne oppintegreringen på, som vi allerede har sett. Tiden er nå moden for å se på kodingen av de numeriske metodene.

En generell arbeidsgang ved løsning av numeriske oppgaver kan forsøksvis se slik ut:

- Før du setter deg til datamaskinen, må du ha etablert differensialligningen som beskriver prosessen du er interessert i. Tenk gjennom hvordan du kan finne/bestemme alle størrelser som inngår i differentalligningen. Bestem deg for initialverdier.
- Gå så til datamaskinen ...
- Skriv inn programmet: Definer alle variable du trenger og gi dem deres verdier, blant annet oppløsningen (N) du skal bruke i oppintegreringen.
- Nullstill arrays, eller gi arrays verdier.
- Slå sammen alle uttrykk av faste konstanter som skal brukes i hovedløkken (se nedenfor), slik at du ikke får flere regneoperasjoner enn nødvendig i løkken.
- Angi initialbetingelser.
- Gå inn i en løkke der du foretar oppintegreringen av differensialligningen, f.eks. ved hjelp av fjerde ordens Runge-Kutta.
- Bearbeid dataene videre dersom det er ønskelig.
- Plot resultatene eller presenter dem på annet vis. Lagre data på fil dersom det er ønskelig.

- Sjekk at programmet gir korrekt resultat for en forenklet versjon av problemet hvor det også eksisterer en analytisk løsning. Dette er viktig!
- Gjenta beregningene med ulik oppløsning (ofte gitt ved Δt) for å se hvor mange punkter som trengs for å få god overensstemmelse med det analytiske svaret, eller at resultatet i liten grad avhenger av moderate endringer i oppløsning.
- Tenk minst to ganger gjennom hva du har gjort i beregningene og hvilke resultater du har fått, og forsøk å oppdage faktorer som kan ha ødelagt for kvaliteten i beregningene.
- Tenk også gjennom om den presentasjonsformen du har valgt er tilstrekkelig for det som ønskes studert i beregningene. Ofte er enkle plot ok, men vi kan sjelden lese ut nøyaktige detaljer fra et plot, i alle fall ikke uten at vi har valgt helt spesielle plot som egner seg akkurat for det vi vil vise.(Eksempel: Enkelte ganger er valg av lineære eller logaritmiske akser i plot avgjørende for om du oppdager interessante sammenhenger eller ikke.)
- Når du mener at programmet fungerer slik det skal, kan du endelig tenke på å gjøre de beregningene som skal inn i det prosjektet du arbeider med. Da må krav om reproduserbarhet etterleves ved at programmet nå får et høytidelig versjonsnummer, og det legges inn utskriftsrutiner som sørger for at alle parametre som er brukt dokumenteres gjennom programkoden (som må lagres og ikke endres uten at det blir nytt versjonsnummer) og/eller i resultatfilen for kjøringene som skal gjøres.
- Gjennomfør så beregningene som skal brukes videre i ditt arbeid.
- Filer som dokumenterer kjøringene for ettertiden må tas vare på på tilsvarende måte som en laboratoriejournal.

Mens du driver med programutviklingen og testingen er det viktig å lagre programmet flere ganger underveis, og helst skifte navn noen ganger i tilfelle noe katastrofalt skjer. Da slipper du å måtte starte helt fra nytt av om du mister alt i en fil. Det er også lurt å teste ut *deler* av programmet underveis når det lar seg gjøre.

Får du ikke programmet til å virke slik det skal, må du forsøke å teste ut bit for bit av programmet i den rekkefølgen beregningene gjennomføres. Forsøke å teste mellomverdier (f.eks. skrive dem ut til skjerm). Lag gjerne forenklinger i noen av uttrykkene for å se hvor ting skjærer seg. Det er møysommelig og ofte fryktelig tidkrevende å finne feil. Av den grunn er tipset ovenfor uhyre viktig, nemlig å teste ut bit for bit underveis mens du programmerer. Det dummeste du kan gjøre er å forsøke å skrive hele programmet på en gang, uten noe som helst testing underveis!

3.11 Diverse tillegg

3.11.1 Oppsummering, kapittel 3

La oss forsøke oss med en oppsummering av viktige punkter i vårt kapittel:

- En annen ordens differensialligning kan anses som ekvivalent med to koblede første ordens differensialligninger.
- I en enkel differensialligning kan vi tilnærmet erstatte den deriverte df/dt med differentialet $\Delta f/\Delta t$. Ved å ta utgangspunkt i denne tilnærmede ligningen og initialbetingelsene, kan vi suksessivt regne oss fram til alle senere verdier av $f(t)$. Denne metoden kalles Eulers metode. Metoden gir ofte store feil, spesielt når vi har med svingninger å gjøre!
- Det finnes bedre metoder for å estimere gjennomsnittlig stigningstall for funksjonen i intervallet Δt enn å bare bruke den deriverte i begynnelsen av intervallet slik vi gjør ved Eulers metode. En av de mest praktiske og robuste metodene kalles Runge-Kuttas metode av fjerde orden. I denne metoden benyttes et veiet gjennomsnitt av fire ulike beregnede stigningstall i intervallet Δt som utgangspunkt for beregningene. Metoden gir ofte god overensstemmelse med analytiske løsninger der disse finnes, også for svingefenomener. Vi må imidlertid være klar over at også denne metoden har feil, og for enkelte systemer vil den ikke fungere tilfredsstillende.
- For annen ordens ordinære differensialligninger så som svingeligningen, kan vi finne løsningen såfremt vi kjenner differensialligningen og initialbetingelsene. For annen ordens partielle differensialligninger, for eksempel en bølgeligning, må vi *i tillegg* kjenne de såkalte randbetingelsene, ikke bare ved starten, men også hele tiden underveis i beregningene. Dette gjør at det ofte er langt vanskeligere å løse partielle differensialligninger enn ordinære annen ordens diffligninger.
- Det er verdifullt å sammenligne numeriske beregninger og analytiske beregninger (der disse finnes) for å oppdage feil i programeringen vår. Men selv om overensstemmelsen er god i slike spesialtilfeller, er det ingen garanti for at løsningene er korrekte også for andre parametre (der analytiske løsninger ikke finnes).
- Iblant skrives en differensialligning om slik at bare "dimensjonsløse variable" benyttes ved den numeriske løsningen. En viktig grunn er å redusere faren for tap av numerisk presisjon. I vår sammenheng er dette ikke så viktig, blant annet fordi vi arbeider med fenomener der lengdeskalaen ikke er svært langt fra en meter, og tidsskalaen ikke er svært forskjellig fra et sekund.
- Siden vi lett kan endre på programmer og parametre, er det en stor utfordring å holde rede på hvordan dataprogrammet så ut og hvilke parametre vi brukte da vi foretok beregninger og kom fram til resultater vi vil anvende. En eller annen systematisk form for dokumentasjon er tvingende nødvendig, der program, input parametre og resultater kan kobles mot hverandre på en entydig måte.

3.11.2 Pseudokode for Runge-Kuttas metode *

```
Funksjonen tar inn x[n-1], v[n-1] og t[n-1] og returnerer x[n] og v[n].

1. Bruk innparameterene til å finne akselerasjonen, a1, i begynnelsen av intervallet.
   Farten i starten av intervallet, v1, er gitt som innparameter.
   x1 = x[n-1]
   v1 = v[n-1]
   a1 = ...

2. Bruk denne akselerasjonen og farten til å finne et estimat for farten (v2) og
   posisjonen i midten av intervallet.
   x2 = ...
   v2 = ...

3. Bruk den nye posisjonen og farten for å finne et estimat for akselerasjonen,
   a2, i midten av intervallet.
   a2 = ...

4. Bruk nå den nye akselerasjonen og farten (a2 og v2) til å finne et nytt estimat
   for posisjonen og farten (v3) i midten av intervallet.
   x3 = ...
   v3 = ...

5. Bruk deretter den nye posisjonen, farten og tiden i midten av intervallet til å
   finne et nytt estimat for akselerasjonen, a3, i midten av intervallet.
   a3 = ...

6. Benytt så det siste estimatet for akselerasjonen og farten i midten av
   intervallet for å finne et estimat for posisjonen og farten (v4) i slutten av
   intervallet.
   x4 = ...
   v4 = ...

7. Bruk så det siste estimatet for posisjonen og farten for å finne et estimat
   for akselerasjonen i slutten av intervallet, a4.
   a4 = ...

8. En middelverdi for farten og akselerasjonen i intervallet beregnes så ved
   hjelp av et vektet gjennomsnitt:
   vMiddle = 1.0/6.0 * (v1 + 2*v2 + 2*v3 + v4)
   aMmiddle = 1.0/6.0 * (a1 + 2*a2 + 2*a3 + a4)

9. Til slutt brukes denne midlede verdien for farten og akselerasjonen i intervallet
   til å beregne posisjonen og farten i slutten av intervallet. Funksjonen returnerer
   denne farten og posisjonen.
   x[n]  = ...
   v[n] = ...
   return x[n], v[n]
```

3.11.3 Python-kode for Runge-Kuttas metode

```
# Først en egen selvstendig funksjon som på basis av kun ETT punkt (x,v,t) kan
# angi hvordan differensialligningen ser ut akkurat i dette punktet.
# Svaret returneres og brukes av den generelle Runge-Kutta funksjonen.
# MERK: Det er KUN denne lille funksjonen som endres når vi går
# fra ett system til et annet. Den generelle RK-funksjonen er identisk
# for alle annen ordens diffligninger vi skal bruke.
def diffEq(xNow,vNow,tNow):
    aNow = f(xNow,vNow,tNow)
    # Her må vi erstatte linjen foran
    # med den differensialligningen
    # vi skal løse numerisk.
    return aNow

# Funksjon som tar inn startpunktet og  bruker Runge-Kuttas metode for å finne
# neste punkt. Kaller på funksjonen diffEq over.
def rk(xStart,vStart,tStart):
    a1 = diffEq(xStart,vStart,tStart)
    v1 = vStart

    xHalf1 = xStart + v1 * dt/2.0
    vHalf1 = vStart + a1 * dt/2.0

    a2 = diffEq(xHalf1,vHalf1,tStart+dt/2.0)
    v2 = vHalf1

    xHalf2 = xStart + v2 * dt/2.0
    vHalf2 = vStart + a2 * dt/2.0

    a3 = diffEq(xHalf2,vHalf2,tStart+dt/2.0)
    v3 = vHalf2

    xEnd = xStart + v3 * dt
    vEnd = vStart + a3 * dt

    a4 = diffEq(xEnd,vEnd,tStart + dt)
    v4 = vEnd

    aMiddle = 1.0/6.0 * (a1 + 2*a2 + 2*a3 + a4)
    vMiddle = 1.0/6.0 * (v1 + 2*v2 + 2*v3 + v4)

    xEnd = xStart + vMiddle * dt
    vEnd = vStart + aMiddle * dt

    return xEnd, vEnd
```

3.11.4 Matlab-kode for Runge-Kuttas metode

```
function [xp,vp,tp] = rk4r(xn,vn,tn,delta_t,param)

% Runge-Kutta integrator (4. orden) Versjon 09022013.
%*************************************************************
% Denne versjonen av fjerde ordens Runge-Kutta rutine for Matlab
% er skrevet av Arnt Inge Vistnes for bruk i FYS2130.
% Rutinen passer for det tilfellet at vi har to koblede diffligninger
%    dv/dt = ffa(x,v,t,param)
%    dx/dt = v
% x,v,t kan være hhv posisjon, hastighet og tid.  delta_t er steplengden i tid.
% param er ulike parametre som inngår i diffligningene, blant annet MÅ
% navnet på funksjonen som gir annenderivert angis i param.
% Dette er en funksjon som brukeren selv må sette opp.
% Input argumenter (n: "nå")
%   [xn,vn,tn] = nåværende verdier for x, v og t.
% Output argumenter (p : "n plus 1")
%   [xp,vp,tp] = nye verdier for x, v og t etter et step i delta_t.
%*************************************************************

ffa = eval(['$\@$' param.fn]);   % Henter opp navn på Matlabkode for annenderivert

halv_delta_t = 0.5*delta_t;
t_p_halv = tn + halv_delta_t;

x1 = xn;
v1 = vn;
a1 = ffa(x1,v1,tn,param);

x2 = x1 + v1*halv_delta_t;
v2 = v1 + a1*halv_delta_t;
a2 = ffa(x2,v2,t_p_halv,param);

x3 = x1 + v2*halv_delta_t;
v3 = v1 + a2*halv_delta_t;
a3 = ffa(x3,v3,t_p_halv,param);

tp = tn + delta_t;
x4 = x1 + v3*delta_t;
v4 = v1 + a3*delta_t;
a4 = ffa(x4,v4,tp,param);

% Returnerer (tilnærmet) (x,v,t) i slutten av intervallet.
delta_t6 = delta_t/6.0;
xp = xn + delta_t6*(v1 + 2.0*(v2+v3) + v4);
vp = vn + delta_t6*(a1 + 2.0*(a2+a3) + a4);

tp = tn + delta_t;
return;
```

3.11.5 Funksjonen som inneholder differensiallikningen

```
function dvdt = tvungen(y,v,t,param)

%*********************************************************
% Funksjon for bruk i FYS2130.  Versjon 09022013
% Returnerer venstresiden i en diffligning for dv/dt
% Benyttes av Runge-Kutta 4 numerisk løsning av to koblede
% diffligninger, f.eks. for svingebevegelse for ulike varianter forhold.
% Input argumenter ("n" indikerer "nå")
%   xn = posisjon
%   vn = hastighet
%   tn = tid
% Output argument
%   dvdt = Venstresiden av første ordens diffligningen for v
% MERK: Dette er en funksjon der den påtrykte kraften på en fjærpendel bare
% varer fram til en viss tid (param.slutt) overført som del av "param"-strukturen.
% Også de øvrige parametrene for fjærkonstant, masse etc overføres i param.
% Se hovedprogrammet for å se hvordan param defineres!!!
%*********************************************************

% Tvungen harmonisk svingning hvor den påtrykte kraften slutter etter en stund
if (t < param.slutt)
    dvdt = - param.A*v - param.B*y + param.C*cos(param.D*t);
else
    dvdt = - param.A*v - param.B*y;
end;
return;
```

3.11.6 Eksempel: Matlabprogram som bruker Runge-Kutta

Nedenfor er det gitt et program for beregning av tvungne mekaniske svingninger (fjærpendel). Det viser hvordan Runge-Kutta brukes i praksis dersom vi programmerer Runge-Kutta rutinen selv. I programmet er det lagt inn et eksempel på kode som vil kunne dokumentere hvilket program og hvilke parametre som ligger bak en konkret beregning. En slik dokumentasjon er nødvendig for å tilfredsstille dagens krav til reproduserbarhet.

```
function tvungSving14

% Eksempelprogram for å undersøke tvungne svingninger som starter
% (initialbetingelser: systemet i ro), og den påtrykte kraften kuttes ut etter en stund.
% Programmet er brukt i kurset FYS2130 ved UiO våren 2014. Skrevet av Arnt Inge Vistnes.
% Programmet kaller på funksjonene rk4r.m og tvungen.m

global param;

% Konstanter m.m.
omega = 100;
```

```
Q = 25;
m = 1.0e-2;
k = m*omega*omega;
b = m*omega/Q;
F = 40;
tid = 6.0;
% Parametre som brukes i selv beregningene (rk4.m, tvungen.m)
param.A = b/m;
param.B = omega*omega;
param.C = F/m;
param.D = omega*1.0;  % Vinkelfrekvens til påtrykt kraft
param.slutt = tid/2.0;
param.fn = 'tvungen'; % Navn på Matlabfil for annen derivert

% Valg av antall step og stepstørrelse i beregningene
N = 2e4;                    % Antall beregningspunkter
delta_t = tid/N;           % Tidssteg i beregningene

% Allokering av arrayer, initialbetingelsene angis
y = zeros(1,N);
v = zeros(1,N);
t = zeros(1,N);
y(1) = 0.0;
v(1) = 0.0;
t(1) = 0.0;

% Løkken hvor beregninger blir foretatt
for j = 1:N
    [y(j+1), v(j+1), t(j+1)]=rk4r(y(j),v(j),t(j),delta_t,param);
end;

% Plotting av resultater
plot(t,y,'-b');
maks = max(y);
xlabel('Tid (rel enhet)');
ylabel('Momentant utslag (rel enhet)');
axis([-0.2   tid   -maks*1.2   maks*1.2]);

return
```

Dersom vi ønsket å skrive parametre til fil for dokumentasjon av hva som lå bak en beregning, kunne følgende kode legges til i programmet ovenfor (eller som en egen funksjon):

```
% Dokumentasjon på kjøring (for lagring)
filnavn = 'tvungSvingBeregn09022013.txt';
dato = '09022013';
fileID = fopen(filnavn, 'w');
fprintf(fileID,'Kjøring %s av programmet "tvungSving14",', dato);
fprintf(fileID,'omega %d \r\n', omega);
fprintf(fileID,'Q %d \r\n', Q);
fprintf(fileID,'Massen m %f \r\n', m);
```

```
fprintf(fileID,'Fjærstivhet k %f \r\n', k);
fprintf(fileID,'Friksjonsparameter b %f \r\n', b);
fprintf(fileID,'Påtrykt kraft Fmax %f \r\n', F);
fprintf(fileID,'omegaD %f \r\n \r\n', param.D);
fprintf(fileID,'Total tid %f \r\n \r\n', tid);
fprintf(fileID,'Tid når kraft opphører %f \r\n \r\n', param.slutt);
fprintf(fileID,'Antall pkt i beregningen %d \r\n \r\n', N);
fprintf(fileID,'Beregnede data t(i), y(i), v(i) \r\n');
for i=1:N
    fprintf(fileID,'%f  %f  %f \r\n', t(i), y(i), v(i));
end;
fclose(fileID);
```

3.11.7 Bruk av Matlab's innebygde Runge-Kutta *

Her følger til slutt et eksempelprogram for beregning av dempede svingninger dersom vi bruker Matlab's innebygde løser av ordinær diffligninger (ode) ved hjelp av 4. ordens Runge-Kutta. Først angir vi hovedprogrammet som vi har kalt *dempetSvingning.m* (navnet er uvesentlig her), og dernest følger en liten programsnutt *vaarDiffLign.m* som hovedprogrammet kaller på. Matlab's ligningsløser krever nemlig en liten ekstra funksjon som angir den aktuelle differensialligningen som sådan, og det er den som gis i vaarDiffLign.m.

```
% Program for å simulere dempede svingninger. Laget 31. januar 2010  med utgangspunkt
% i et program laget av Filip Nicolaisen noen dager før.
% Løser de to koblede differensiallikningene
% dz/dt = v
% dv/dt = - koef1 v - koef2 z

clear all;

% Angir systemets fysiske egenskaper (i SI-enheter)
b = 3.0; % Friksjonstall
m = 7.0; % Massen
k = 73.0; % Fjærstivhet
% Retningslinje:
%    Overkritisk dempning : b > 2 sqrt(k m)
%    Kritisk demping : b = 2 sqrt(k m)
%    Underkritisk demping: b < 2 sqrt(k m)

koef1 = b/m;
koef2 = k/m;

% Initialbetingelser (i SI-enheter)
z0  = 0.40; % Posisjon rel. likevektspunkt
v0  = 2.50; % Hastighet

% Tid vi ønsker å følge systemet i [start, slutt]
TID = [0,20];
```

```
% Initialverdier
INITIAL=[z0,v0];

% Lar Matlab selv gjennomføre en full fjerde ordens Runge-Kutta oppintegrering
% av differensialligningen. Vår konkrete diffligning er spesifisert i funksjonen
% vaarDiffLign.
% T er tiden, F er løsningene [z v], tilsvarende er t den løpende variabelen
% tid og f den løpende variabelen [z(t) v(t)] som Matlab benytter ved beregningene.
% Matlab velger selv hvor tett punktene skal ligge for å gi ok nøyaktighet.
% Punktene er ikke ekvidistante i tid!

[T F] = ode45(@(t,f) vaarDiffLign(t,f,koef1,koef2),TID, INITIAL);

% Plotting av resultatet, velger bare å  plotte posisjon vs tid.
plot(T,F(:,1));

% length(T) % Kan ta med for å se hvor mange punkter Matlab faktisk
% brukte ved løsning av difflign over ditt valgte tidsintervall.

% Her burde det legges inn en test på at vår beregning gir identisk resultat
% med analytisk uttrykk i et tilfelle hvor analytisk uttrykk finnes.
```

Her kommer så den lille funksjonen som gir selve differensialligningen (i form av to koblede differensialligninger):

```
function df = vaarDiffLign(~,f,koef1,koef2)

% Denne funksjonen evaluerer funksjonene f, hvor f(1) = z og f(2)=v.
% Som første variabel i parametre inn har vi skrevet ~ fordi tiden ikke
% inngår eksplisitt i uttrykkene våre.

df = zeros(2,1);

%HER kommer det vesentlige: Den ene diffligningen: dz/dt = v
df(1) = f(2);

% Den andre diffligningen: dv/dt = -koef1 v - koef2 z
df(2) = -koef1*f(2)-koef2*f(1);
```

3.11.8 Noen "kjøreregler" fra Hans Petter Langtangen

Programmering er et eget fag forskjellig fra fysikk. Denne boka er skrevet av meg, en fysiker, som finner programmering svært nyttig for mitt fag, men som ikke har like stor stolthet som en informatiker knyttet til utformingen av de programmene jeg skriver. For meg er det viktigst å være i stand til å foreta numeriske beregninger der analytiske beregninger er umulige eller for kompliserte til å gjennomføre i praksis.

Det betyr at mine dataprogrammer sjeldent ville fått "beste karakter" av en ren informatiker. Til tross for dette har jeg svært mye glede av min programmering.

Du får selv velge hvor elegante dataprogrammer i informatikerens øyne du vil skrive.

Hans Petter Langtangen har gitt noen prinsipper for god programmering. Med utgangspunkt i disse vil jeg nevne:

- Det bør være en-til-en korrespondanse mellom matematisk beskrivelse av et problem (algoritme) og koden. Det gjelder variable, formler osv.
- En kode bør stykkes opp i logiske funksjoner. I Python kan flere funksjoner legges i samme fil. I Matlab legges gjerne ulike funksjoner i separate filer (selv om det faktisk er mulig å bruke et lignende oppsett i Matlab som i Python). Dette fører ofte til at programmering i Matlab gir "flate" programmer, mye hardkoding av formler og mangel på logisk oppdeling av programmet.
- Programmér generelt der det ikke er uhensiktsmessig. For eksempel når man skal integrere et uttrykk, programmeres et generelt integral av f(x) og så sender man inn sin spesielle f som argument. Dette krever hyppig bruk av funksjoner.
- Bruk energi på å konstruere testproblemer for å sjekke at implementasjonen er riktig.

3.11.9 Forslag til videre lesing

Følgende kilder være nyttige dersom du ønsker å gå litt dypere i dette stoffet:

- Hans Petter Langtangen: A primer on scientific programmiinng with Python. 3rd Ed. Springer, 2012.
- http://en.wikipedia.org/wiki/Semi-implicit_Euler_method (tilgjengelig 13.01.2013)
- `http://en.wikipedia.org/wiki/Numerical_partial_differential_equations`

  (tilgjengelig 13.01.2013)
- "Kalkulus", 3. utgave av Tom Lindstrøm (Universitetsforlaget).

3.11.10 Et annet lite tips....

Du vil antakelig lagre en masse plot etter dine beregninger i dette og andre kurs. Plottene legges så inn i obliger, rapporter, og eventuelt senere i masteroppgaver og denslags. Mange studenter gjør da noe ganske dumt.

Det er et krav at tall og tekst langs aksene på plottene må være godt leselige uten bruk av lupe (!) *i den endelige størrelsen figurene har i et dokument.* Det betyr at tall og bokstaver bør ha en størrelse på mellom 9 og 12 pt i *endelig størrelse* (kan gå ned til 8 pt om sterkt ønskelig, og indekser kan til nød være enda litt mindre).

Det kan være lurt å lagre figurer i Matlab mens figurene *ikke* har fylt hele skjermen (bruk default display av figurer på skjermen). Da blir fontstørrelsen ofte tilstrekkelig stor selv om figuren forminskes omtrent til samme format som er brukt i denne boka. Reduseres imidlertid figurstørrelsen for mye, vil fontstørrelsen i det endelige dokumentet blir for liten. Du kan selv velge fontstørrele i plot som genereres av Matlab og Python.

Lær deg gode vaner så raskt som mulig, - det vil lønne seg i det lange løp!

3.12 Læringsmål

Etter å ha jobbet deg gjennom dette kapittelet bør du ...

- kjenne til at en annen ordens differensialligning kan anses som ekvivalent med to koblede første ordens differensialligning.
- kunne løse en annen ordens differensialligning numerisk ved hjelp av fjerde ordens Runge-Kuttas metode.
- kunne forklare hvorfor numeriske mye oftere enn analytiske løsningsmetoder kan behandle kompliserte fysiske lovmessigheter, f.eks. ikke-lineær friksjon.
- kunne peke på noen faktorer som kan føre til at numeriske beregninger fungerer dårlig.
- kunne forklare i grove trekk hvorfor fjerde ordens Runge-Kutta vanligvis fungerer bedre enn Eulers metode.
- kunne foreta en rimelig god test på at et dataprogram som bruker numeriske løsningsmetoder fungerer som det skal.
- kjenne til hvordan vi kan gå fram for å sikre dokumentasjon av program og parametre som hører sammen med beregnede verdier.
- kjenne til hvorfor det er lurt å lagre et dataprogram under nye navn rett som det er mens man driver programutviklingen.
- kjenne til noen prinsipper som bør anvendes for å unngå omfattende feilsøking "til slutt", og kjenne noen triks som kan brukes ved feilsøking av programmer.

3.13 Oppgaver

Forståelses- / diskusjonsspørsmål

1. Hvorfor fungerer fjerde ordens Runge-Kutta vanligvis bedre enn Eulers metode?

2. Ved numerisk løsning av en svingeligning, trenger vi å kjenne systemets karakteristiske egenskaper, og vi må kjenne initialbetingelsene. Angi karakteristiske egenskaper for en mekanisk fjær-pendel og for en RCL-svingekrets. Angi typiske initialbetingelser for de samme systemene.

3. I kapitlet om svingninger fortalte vi litt om begrensinger i superposisjonsprinsippet. I dette kapitlet har vi foretatt beregninger hvor superposisjonsprinsippet ikke gjelder. Pek på et system som passer til denne karakteriseringen, og forklar hvorfor superposisjonsprinsippet ikke gjelder. Kan du foreslå hvordan du kan bevise dette?

4. Forsøk å skissere arbeidsoppgaven vi bruker når vi vil beregne et skrått kast med eller uten friksjon (eller planetbevegelse rundt Sola) analytisk. Hva bruker vi mest tid på, og hva konsentrerer vi oss om når vi betrakter beregningen i ettertid? Forsøk å skissere arbeidsoppgaver og hvordan vi betrakter resultatet ved numeriske beregninger. Hva er fordeler og ulemper med analytisk matematikk, sammenlignet med numeriske metoder? Forsøk også å trekke inn fysisk forståelse av mekanismene for bevegelsene.

Regneoppgaver

5. Lag ditt eget program for å beregne tidsutviklingen til en dempet svingning ved hjelp av fjerde ordens Runge-Kutta løsningsmetode. Test at den fungerer ved å sammenligne resultatene for analytisk løsning og numerisk løsning i det tilfellet de skal være identiske. Hvor stor er feilen i posisjonen for den numeriske løsningen (relativt til max utslag)? Dersom du selv velger tidssteget Δt ber vi deg å teste ut minst to-tre ulike valg for Δt for å se hvor mye dette valget betyr for nøyaktigheten.

6. Vi utledet ligning (3.10) ved å ta utgangspunkt i Taylorutvikling. I teksten like etter ligningen antyder vi at uttrykket kan utledes også på en annen måte. Gjennomfør denne utledningen.

7. Gjennomfør beregninger av tvungne svingninger for en rekke forskjellige påtrykte frekvenser, og sjekk at uttrykket for kvalitetsfaktor i kapittel 1 stemmer overens med frekvenskurven og den alternative beregningen av Q basert på halvverdibredde og senterfrekvens.

8. Studér hvor raskt amplituden vokser ved tvungne svingninger når den påtrykte frekvensen er litt forskjellig fra resonansfrekvensen. Sammenlign med tidsforløpet ved resonansfrekvensen. Initialbetingelser: Systemet starter i ro fra likevektspunktet.

9. Finn ut hvordan beregningene i de forgående oppgavene måtte modifiseres dersom vi f.eks. ønsket å innlemme et ekstra ledd $-cv^2\frac{\vec{v}}{v}$ for friksjonen. Kommenter gjerne din oppfatning av hvorfor numeriske metoder har en del fortrinn framfor analytiske matematiske metoder alene.

10. Denne oppgaven går ut på å sjekke om superposisjon gjelder for et svingende fjærpendelsystem med demping, først i det tilfellet at friksjonen kan beskrives kun med et ledd av typen $-bv$, dernest i det tilfellet at friksjonen må beskrives ved $-bv - sv^2$, eller rettere sagt: $-bv - s|v|v$ for å ta hensyn til retningen (se kapittel 1 hvor denne detaljen er nevnt). Rent praktisk innebærer oppgaven at du må gjøre beregninger for én svingetilstand, dernest for en annen, og så sjekke om summen av løsninger er lik løsningen av summen av tilstander.

    Fjærpendelens fysiske egenskaper er karakterisert ved $b = 2.0$, $s = 4.0$, $m = 8.0$ og $k = 73.0$, alt i SI-enheter. Gjør beregninger først med initialbetingelsene $z0 = 0.40$ og $v0 = 2.50$, og dernest for initialbetingelsene $z0 = 0.40$ og $v0 = -2.50$. Legg sammen de to løsningene. Sammenlign denne med løsningen av differensialligningen når initialbetingelsene er lik summen av initialbetingelsene vi brukte i de to første kjøringene. Husk at du skal sjekke superposisjonsprinsippet både for kjøringer hvor $-s|v|v$-leddet er med og der det ikke er med. Kan du trekke en foreløpig konklusjon / fremsette en hypotese om gyldigheten til superposisjonsprinsippet ut fra resultatene du har kommet til?

    [♠ ⇒ NB: I tilfelle du bruker Matlabs innebygde løser, vil tidspunktene ikke stemme overens mellom de to kjøringene. Du må da ta utgangspunkt i tidsrekken svarende til den ene kjøringen og bruke interpolasjon når addisjon av resultat for den andre kjøringen skal gjennomføres. Nedenfor er det gitt et eksempel på hvordan en slik addisjon kan foretas. Spør gruppelærer dersom du ikke forstår koden godt nok til å bruke den eller noe lignende i eget program.

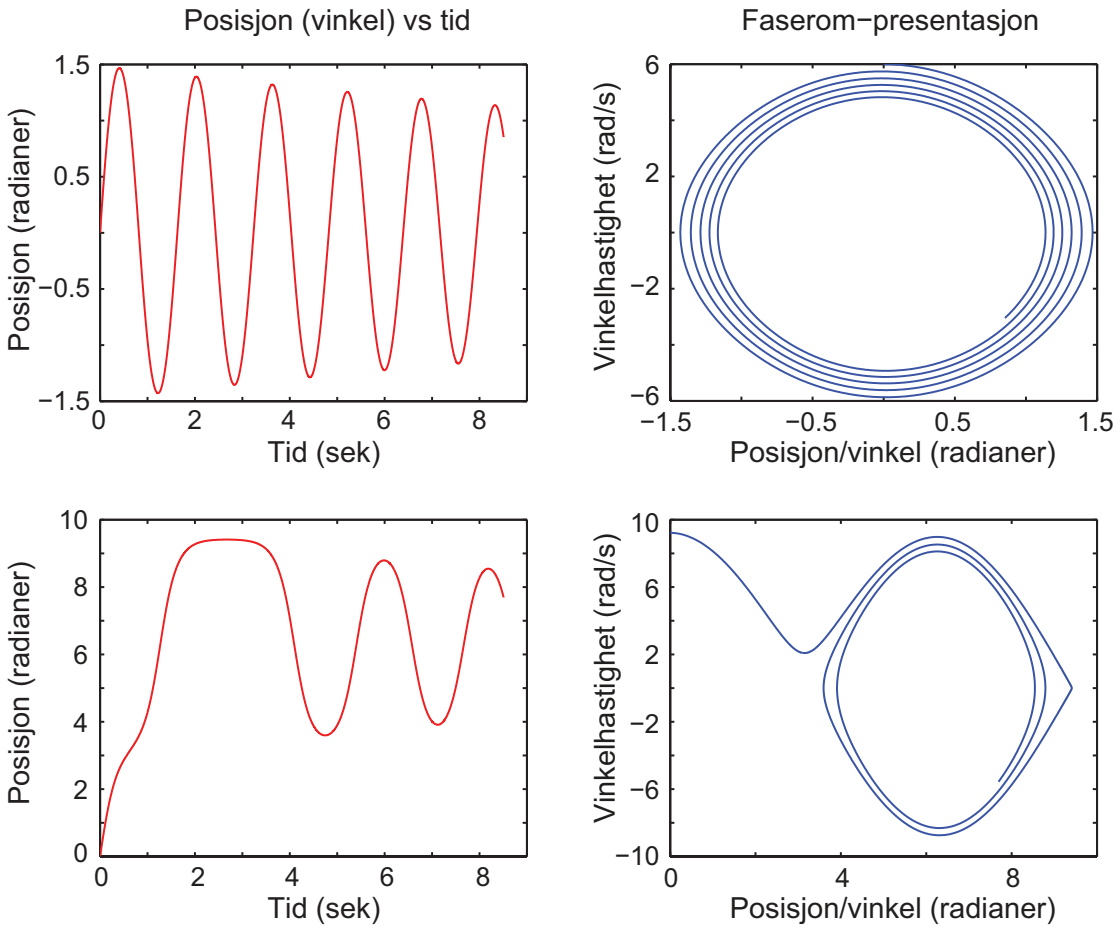
```
% Addisjon av to funksjoner Z1(t) og Z2(t'), hvor t er elementer i T1 og t' i T2.
% De to settene har samme startverdi (og sluttverdi), men ellers ulike.
% n1 = length(T1) og n2 = length(T2). Rutinen virker bare dersom n2>=n1.
% I motsatt fall må koden justeres tilsvarende.

% Tar T1 som basis for summasjonen
Z12(1)=Z1(1)+Z2(1);
for i = 2:n1
    % Finner først indeks til siste punkt i T2 mindre enn T1(i)
    j = 1;
    kL = -1;
    while kL<0
        if (T2(j)<T1(i)) j=j+1;
        else;
            kL=j-1;
        end;
    end;
    % Da har første punkt i T2 større eller lik T1(i) indeksen:
    kH = kL+1;
```

```
    % Summerer de to løsningene (lineær interpolasjon)
    Z12(i) = Z1(i)+Z2(kL) + (Z2(kH)-Z2(kL))...
    *(T1(i)-T2(kL))/(T2(kH)-T2(kL));
end;
```

⇐ ♠]

11. I figur 3.6 er resultatet av beregninger av en pendelbevegelse vist for tilfellet at det er litt friksjon til stede. Figuren viser posisjon (vinkel) som funksjon av tid (venstre del) og vinkelhastighet som funksjon av posisjon (vinkel) i høyre del (også kalt fasediagram). De to øvre figurene fremkommer ved en initialbetingelse der pendelen ved tiden $t = 0$ henger rett ned, men samtidig har en liten vinkelhastighet. De nedre figurene fremkommer ved at initialbetingelsene er som for øvre del, men at den initielle vinkelhastigheten er en god del større enn i det første tilfellet.

    Forklar hva figurene sier om bevegelsene (forsøk å få med så mange interessante detaljer som mulig). Hvordan ville figuren sett ut dersom vi økte den initielle vinkelhastigheten enda litt mer enn den vi har i nedre del av figuren?



Figur 3.6: *Bevegelsen til en enkel pendel. Se teksten for omtale.*

3.13.1 En mer sammensatt regneoppgave

I denne oppgaven skal du gjennomgå en løsning av en annen ordens inhomogen differensialligning ved hjelp av et spesielt dataprogram som gir analytiske uttrykk for løsningen, og dernest skal du gjøre egne beregninger ut fra numeriske metoder og sammenligne resultatet. Underveis gis det en rekke spørsmål om detaljer som tester forståelsen av hva som foregår.

12. I de to kapitlene om svingninger ble det vist hvordan vi kan løse den andre ordens differensialligningen som beskriver strømmen i en RCL-krets med eller uten ytre påtrykt vekselspenning. Det var lett å finne løsningen av den homogene differensialligningen (uten påtrykt spenning), men litt mer styr for å finne en partikulær løsning av den inhomogene ligningen (som svarer til påtrykt ytre vekselspenning). Så lenge den påtrykte spenningen var et rent sinusignal hvor verken frekvens eller amplitude endret seg med tiden, var det likevel en overkommelig oppgave.

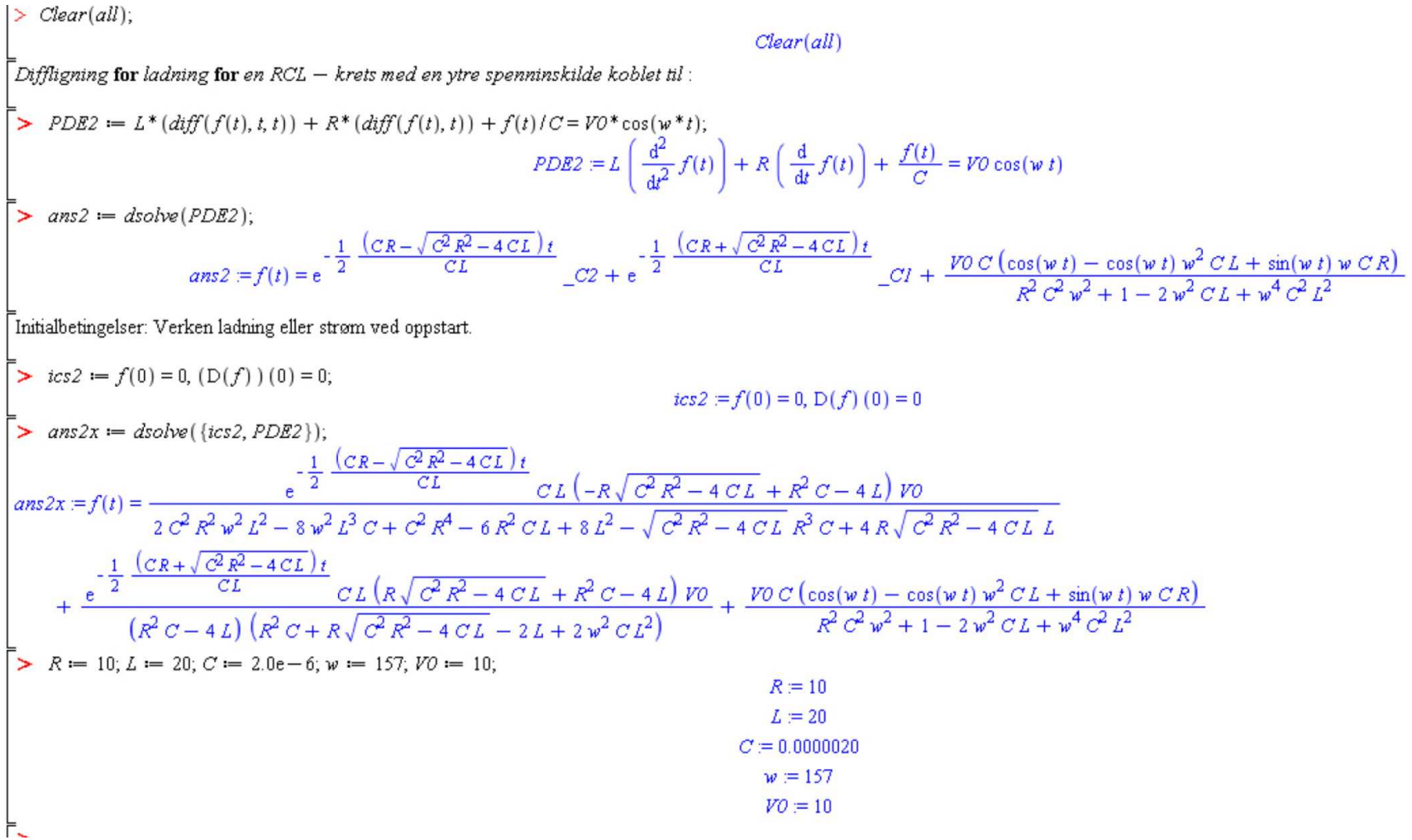
    Så lenge vi opererer bare med generelle løsninger, inngår det et par koeffisienter i uttrykkene vi kommer fram til. Da ser løsningen rimelig grei ut. Dersom vi imidlertid skal finne en bestemt løsning ut fra et sett initialbetingelser, får ofte de ellers valgfrie koeffisientene et nokså komplisert uttrykk. Dette vil du få se et eksempel på i denne oppgaven.

    Det finnes dataprogrammer som kan foreta analytiske beregninger i matematikken i mange tilfeller der dette lar seg gjøre. De mest kjente programmene av denne typen er Matematica og Maple. Du har muligens tilgang til Maple eller Matematica, og vi har derfor valgt å vise et eksempel på hvordan et slikt program kan brukes.

    Figur 3.7 og 3.8 viser en kjøring i Maple hvor vi løser den inhomogene differensialligningen som beskriver hvordan ladningen på en kondensator varierer med tiden i en RCL-krets når vi har en påtrykt spenning med fast vinkelfrekvens. Først får vi en generell løsning. Dernest angis initialbetingelser, og Maple gir oss den spesielle løsningen for disse initialbetingelsene. Til slutt setter vi inn et sett verdier for resistans, kapasitans og induktans, såvel som amplitude og vinkelfrekvens på den påtrykte spenningen, og og ser hvordan ladningen da varierer som funksjon av tid.

    De ivrigste av dere kan forsøke å starte Maple selv og teste ut de kommandoene som er brukt.

    Studer figur 3.7 og forsøk å gjenkjenne kommandoer ut fra de differensialligningene vi kjenner fra kapittel 1. Merk at det vi selv gir som input til Maple er angitt med sort, mens responsen fra Maple er gitt i blått. Derivering kan angis på to måter i Maple, både som "$diff(f(t),t)$" og som "$D(f)(t)$" for enkeltderivert av en funksjon f (som i vårt tilfelle er ladningen q på kondensatoren).



> Clear(all);

$$Clear(all)$$

Diffligning **for** *ladning* **for** *en RCL − krets med en ytre spenninskilde koblet til* :

> $PDE2 := L*(diff(f(t),t,t)) + R*(diff(f(t),t)) + f(t)/C = V0*\cos(w*t);$

$$PDE2 := L\left(\frac{d^2}{dt^2}f(t)\right) + R\left(\frac{d}{dt}f(t)\right) + \frac{f(t)}{C} = V0\cos(w\,t)$$

> $ans2 := dsolve(PDE2);$

$$ans2 := f(t) = e^{-\frac{1}{2}\frac{\left(CR-\sqrt{C^2R^2-4CL}\right)t}{CL}}\_C2 + e^{-\frac{1}{2}\frac{\left(CR+\sqrt{C^2R^2-4CL}\right)t}{CL}}\_C1 + \frac{V0\,C\left(\cos(w\,t) - \cos(w\,t)\,w^2\,C\,L + \sin(w\,t)\,w\,C\,R\right)}{R^2\,C^2\,w^2 + 1 - 2\,w^2\,C\,L + w^4\,C^2\,L^2}$$

Initialbetingelser: Verken ladning eller strøm ved oppstart.

> $ics2 := f(0) = 0, (D(f))(0) = 0;$

$$ics2 := f(0) = 0, D(f)(0) = 0$$

> $ans2x := dsolve(\{ics2, PDE2\});$

$$ans2x := f(t) = \frac{e^{-\frac{1}{2}\frac{\left(CR-\sqrt{C^2R^2-4CL}\right)t}{CL}}\,CL\left(-R\sqrt{C^2R^2-4CL} + R^2C - 4L\right)V0}{2\,C^2R^2w^2L^2 - 8w^2L^3C + C^2R^4 - 6R^2CL + 8L^2 - \sqrt{C^2R^2-4CL}\,R^3C + 4R\sqrt{C^2R^2-4CL}\,L}$$
$$+ \frac{e^{-\frac{1}{2}\frac{\left(CR+\sqrt{C^2R^2-4CL}\right)t}{CL}}\,CL\left(R\sqrt{C^2R^2-4CL} + R^2C - 4L\right)V0}{\left(R^2C-4L\right)\left(R^2C + R\sqrt{C^2R^2-4CL} - 2L + 2w^2CL^2\right)} + \frac{V0\,C\left(\cos(w\,t) - \cos(w\,t)\,w^2\,C\,L + \sin(w\,t)\,w\,C\,R\right)}{R^2\,C^2\,w^2 + 1 - 2\,w^2\,C\,L + w^4\,C^2\,L^2}$$

> $R := 10; L := 20; C := 2.0e-6; w := 157; V0 := 10;$

$$R := 10$$
$$L := 20$$
$$C := 0.0000020$$
$$w := 157$$
$$V0 := 10$$

Figur 3.7: *Eksempel på løsning av en differensialligning i dataprogrammet Maple. Del 1.*

Figur 3.8: *Eksempel på løsning av en differensialligning i dataprogrammet Maple. Del 2.*

Konkrete oppgaver (refererer ofte til figurene 3.7 og 3.8):

a. Skriv ned differensialligningen som er utgangspunktet for Maple-beregningene (angi den i vår vanlige matematiske språkdrakt).

b. Hvordan angis en differensialligning i Maple, og hvordan løses den?

c. Kjenner du igjen løsningen av differensialligningen ut fra den generelle løsningsprosedyren angitt i kapittel 1 og 2?

d. Hvilke symboler bruker Maple på de to valgfrie koeffisientene i den generelle løsningene av diffligningene?

e. Hvilke initialbetingelser har vi valgt i eksemplet vårt?

f. Beskriv kort hvordan kompleksiteten i uttrykket for løsningen endret seg idet vi måtte ta hensyn til initialbetingelsene. Kan du gi en forklaring på hvorfor endringene gikk i den retning de faktisk gikk?

g. Beregn systemets naturlige svingefrekvens for de valgte verdiene for R, L og C. Hvordan er den valgte påtrykte frekvensen sammenlignet med den beregnede naturlige svingefrekvensen for systemet? (Hint: Husk forskjell mellom frekvens og vinkelfrekvens).

h. Forsøk å angi ut fra plottet nederst i figur 3.8 omtrentlig hvor lenge "innsvingningsforløpet" varer etter oppstart av svingningen med en ytre harmonisk påtrykt spenning. (Merk: Du klarer ikke å se hver enkelt sinussvingning i detalj i plottet fordi linjene ligger for tett. Det fremkommer derfor et indre mønster i kurvene som ikke har basis i virkelig tidsforløp, bare av den begrensede oppløsningen i plottet. Du må bare bruke omhyllingskurven når du gjør estimatet.)

i. Forsøk å angi kvalitetsfaktoren for kretsen bak figur 3.7 og 3.8. (Hint: Ta utgangspunkt i omtalen av kvalitetsfaktoren Q i kapittel 1.)

j. Bruk den vedlagte Matlabkoden (helt til slutt) som bruker Maple-resultatet og regner ut den spesielle løsningen vi har av den inhomogene differensialligningen med de angitte initialbetingelsene. Sjekk at du får samme resultat som angitt i figur 3.8 for de valgte verdier for R, C, L, w og $V0$. Bruk gjerne zoom for å se bedre hvordan tidsforløpet faktisk er.

k. Foreta deretter en systematisk endring i w (frekvensen til påtrykt spenning) for å bestemme omtrentlig Q-verdien til denne kretsen ut fra halvverdibredden for frekvensresponsen (for de angitte verdier for R, C og L). (Hint: Det holder å lese av omtrentlige verdier fra plottene dersom du zoomer inn resultatplottene på en lur måte. Du kan da notere tallverdier som du siden kan bruke for å vise formen til resonanskurven. Fra denne kan du så estimere kvalitetsfaktoren Q).

l. Legg til noen få linjer i Matlabprogrammet for å demonstrere faseforskjell mellom påtrykt spenning og ladningsendringen vs tid. Zoom inn på et egnet område for å

få et kvalitativt bilde av faseskiftet både ved resonansfrekvensen, og litt over og litt under denne. Argumenter for hvor i tidsbildet du gjennomfører sammenligningen.

m. Bruk en numerisk løsningsmetode basert på fjerde ordens Runge-Kutta for å bestemme innsvingningsforløpet under betingelser identiske med de som ligger bak plottet nederst i figur 3.8. Sammenlign resultatet fra kjøringen med det analytiske uttrykket og resultatet fra den numeriske beregningen.

o. Hvordan ville det være å finne en analytisk og numeriske løsning dersom den påtrykte spenningen ikke var en ren sinus?

Her er til slutt Matlab-programmet for simulering av en elektrisk RCL-krets med gitte initialbetingelser:

```
% Beregner tidsforløpet i starten av en svingning i ladning i en RCL-krets
% pådyttet av en ytre vekselspenning med frekvens w og amplitude V0.
% Koden er skrevet med bakgrunn i Maple-beregninger.

R = 10;
L = 20;
C = 2.0e-6;
w = 157*1.00;
V0 = 10;
N = 4000;
t = linspace(0,20,N);

q = zeros(N,1);
rq = zeros(N,1);
partikular = zeros(N,1);

underRottegn = C^2*R^2-4*C*L;
alpha1 = -(1/2)*(C*R-sqrt(C^2*R^2-4*C*L))/(C*L);
alpha2 = -(1/2)*(C*R+sqrt(C^2*R^2-4*C*L))/(C*L);
teller1 = C*L*(-R*sqrt(C^2*R^2-4*C*L)+R^2*C-4*L)*V0;
nevner1 = 2*C^2*R^2*w^2*L^2-8*w^2*L^3*C+C^2*R^4-6*R^2*C*L+8*L^2- ...
sqrt(C^2*R^2-4*C*L)*R^3*C+4*R*sqrt(C^2*R^2-4*C*L)*L;
faktor1 = teller1/nevner1;

teller2 = C*L*(R*sqrt(C^2*R^2-4*C*L)+R^2*C-4*L)*V0;
nevner2 = (R^2*C-4*L)*(R*sqrt(C^2*R^2-4*C*L)+R^2*C-2*L+2*w^2*C*L^2);
faktor2 = teller2/nevner2;
teller3 = C*V0;
nevner3 = R^2*C^2*w^2+1-2*w^2*C*L+w^4*C^2*L^2;
faktor3 = teller3/nevner3;

partikular = cos(w.*t)-cos(w.*t)*w^2*C*L+sin(w.*t)*w*C*R;
q = exp(alpha1.*t).*faktor1 + exp(alpha2.*t).*faktor2 + partikular.*faktor3;

rq = real(q);
plot(t,rq,'-b');
```