KTH Engineering Sciences

Laboration 2

Notera följande:

- **Anmäl er** till en av labbgrupperna SF1546 - **Lab 2** 1–90 (dessa finns långt ner i listan av grupper).
- Egen anmälan till grupp **stänger** måndag den 24 februari.
- Tidsbokningen för muntlig redovisning öppnar tisdag den 25 februari och **stängs** på måndag den 2 mars kl 07.00. Inga anmälningar tas emot efter det.
- Deadline för inlämning av filer är den 1 mars. Se till att filerna är på det format som efterfrågas.

Differentialekvationer

Före redovisningen ska ni skicka in MATLAB*-filer och vissa resultatfiler i Canvas. Vad som ska skickas in står angivet efter respektive uppgift. På redovisningen ska ni (båda i laborationsgruppen om ni är två) kunna redogöra för teori och algoritmer som ni använt. Ni ska också kunna svara på frågepunkterna (•) och förklara hur era* MATLAB*-program fungerar. Kom väl förberedda!*

1. Pendeln

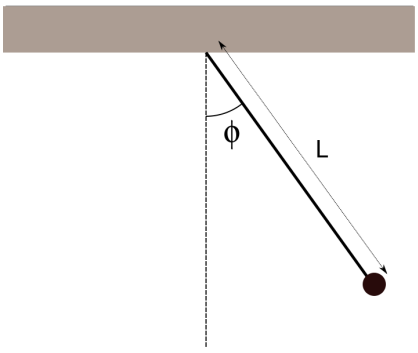
Vi studerar en dämpad pendeln som beskrivs av den ordinära differentialekvationen (ODEn)

$$m\phi'' + \mu\phi' + \frac{mg}{L}\sin(\phi) = 0, \qquad (1)$$

med begynnelsedata

$$\phi(0) = \phi_0, \quad \phi'(0) = 0.$$

Den sökta funktionen $\phi(t)$ är vinkeln för pendelns utslag. Parametrarna i ekvationen är pendelns massa m [kg], dämpningskonstanten μ [N/m], tyngdaccelerationen g [m/s^2] och pendelns längd L [m]. Se figur. I uppgifterna nedan ska ni använda värdena $m = 0.6$, $L = 1.5$, $g = 9.81$ och $\mu = 0.2$.



a) Framåt Euler

Skriv först om (1) till ett system av första ordningens ekvationer. Implementera sedan Framåt Euler-metoden i MATLAB och lös problemet med tidssteget $h = 0.01$ och $\phi_0 = 0.5$ fram till $t = 5$. Plotta lösningen $\phi(t)$. (Hjälp: $0.15 < \phi(5) < 0.25$.) I Canvas står det: Notera att det är det exakta värdet på $\phi(5)$ som ligger i det angivna intervallet. Den numeriska lösningen med $h = 0.01$ hamnar precis utanför intervallet. (Med tillräckligt litet h kommer den dock ligga i intervallet.)

När ni fått detta att fungera ska ni skriva om MATLAB-programmet till en MATLAB-funktion, `feuler` som tar följande indataargument:

- begynnelsedata `u0` (en kolumnvektor med två komponenter),
- sluttiden `T`,
- antal tidssteg `n` (notera: $h = T/n$).

Funktionen ska returnera två variabler:

- kolumnvektorn `t` som innehåller alla tidpunkter, dvs `t = [0,h,2h, ..., T]`^{T} (längd `n+1`),
- matrisen `y` som innehåller två kolumner med approximationerna av ϕ respektive ϕ' i motsvarande tidpunkter (storlek $n + 1$ rader, 2 kolumner).

Notera: Med givna returvärden `t` och `y` från `feuler.m` kan lösningsplotten nu erhållas med kommandot `plot(t,y(:,1))`.

Verifiera att er implementation av Framåt Euler har noggrannhetsordning ett genom att beräkna $\phi(5)$ för flera olika `n` med hjälp av `feuler.m`. Beräkna kvoter av differenser mellan lösningar med n, $2n$, $4n$, $8n$... på samma sätt som i uppgift 4, Laboration 1. Sammanställ en tabell med kvoter och motsvarande noggrannhetsordningar.

Skicka in: Funktionsfilen `feuler.m`; anropas som `"[t,y] = feuler(u0,T,n)"`
(Tabellen skickar ni in nedan.)

b) Runge-Kutta 4

Implementera Runge–Kutta 4 och skapa en funktionsfil med samma argument och returvärden som för Framåt Euler. Verifiera att implementationen är korrekt genom att kolla dess noggrannhetsordning på samma sätt som ni gjorde för Framåt Euler ovan. Lägg till kolumner i tabellen med kvoter och noggrannhetsordningar för Runge–Kutta 4.

- Vad är den teoretiska noggrannhetsordningen?
- Kvantifiera med ett exempel hur mycket noggrannare Runge–Kutta 4 är jämfört med Framåt Euler.
- För små vinklar kan man approximera $\sin\phi \approx \phi$ i (1). Det ger en linjär ODE,

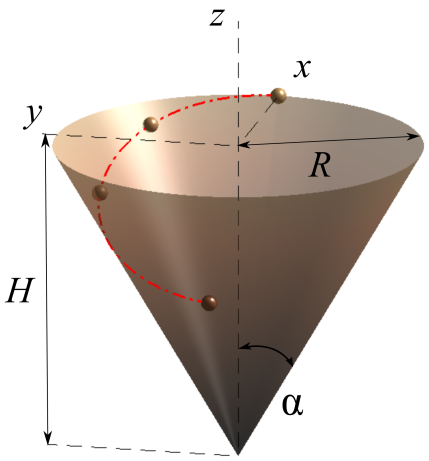
$$m\phi'' + \mu\phi' + \frac{mg}{L}\phi = 0, \qquad \phi(0) = \phi_0, \quad \phi'(0) = 0. \tag{2}$$

  Testa att byta $\sin\phi$ mot bara ϕ i er funktionsfil. Hur mycket ändras lösningarna? Prova med både stora och små begynnelsevärden ϕ_0.

Skicka in: 1) Funktionsfilen `rk4.m`; anropas som "`[t,y] = rk4(u0,T,n)`". 2) Tabellen `kvottabell` med kvoterna och motsvarande noggrannhetsordningar för Framåt Euler och Runge–Kutta 4. Formatet kan vara vanligt textformat eller PDF (men ej `.doc` eller `.odt`).

2. Kula i kon

Betrakta en kula som rör sig inuti en öppen kon med höjden $H = 1.8$ [m], spetsvinkeln $\alpha = \pi/5$ och toppradien $R = H\tan\alpha$ [m]. Kulan ges en horisontell hastighet v_0 [m/s] vid konens övre kant och rullar sedan runt på konens insida. Låt (r, θ, z) vara kulans position i cylinderkoordinater. Dess rörelse beskrivs då av två kopplade andra ordningens ODE för r och θ,



$$r'' = f_1(r, \theta, r', \theta'), \qquad \theta'' = f_2(r, \theta, r', \theta'), \tag{3}$$

där

$$f_1(r, \theta, r', \theta') = -\frac{N}{m}\left(\cos\alpha + \frac{\mu r'}{v}\right) + r\left(\theta'\right)^2,$$
$$f_2(r, \theta, r', \theta') = -\frac{\mu N \theta'}{mv} - \frac{2r'\theta'}{r}.$$

Här är v [m/s] kulans hastighet och N [N] den normalkraft som verkar på kulan, givna av

$$v(r, r', \theta') = \sqrt{(r')^2 + (r\theta')^2 + (r'/\tan\alpha)^2}, \qquad N(r, \theta') = mr(\theta')^2\cos\alpha + mg\sin\alpha. \tag{4}$$

Parametrarna i differentialekvationerna är kulans massa m [kg], tyngdaccelerationen g [m/s^2] och den dimensionslösa friktionskoefficienten μ. Begynnelsedata ges av $r(0) = R$, $\theta(0) = 0$, $r'(0) = 0$ och $\theta'(0) = v_0/R$.

ODEerna (3) kan skrivas om som ett system av fyra första ordningens ODE. Genom att sätta $u_1 = r$, $u_2 = \theta$, $u_3 = r'$ och $u_4 = \theta'$ får vi

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{F}(\boldsymbol{u}), \qquad \boldsymbol{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix}, \qquad \boldsymbol{F}(\boldsymbol{u}) = \begin{pmatrix} u_3 \\ u_4 \\ f_1(u_1, u_2, u_3, u_4) \\ f_2(u_1, u_2, u_3, u_4) \end{pmatrix}.$$

Funktionen $\boldsymbol{F}(\boldsymbol{u})$ finns implementerad i filen `fkula.m` som ligger i Canvas. Parametervalen där är $m = 1$, $g = 9.81$ och $\mu = 0$ (ingen friktion). Observera att funktionen tar två argument, trots att t-parametern inte används. Detta beror på att MATLABs inbyggda ODE-funktioner kräver ett sådant format.

a) Beräkna kulans bana med Runge-Kutta 4

Skriv ett MATLAB-program som beräknar kulans bana med Runge–Kutta 4. Programmet ska anropa funktionen `fkula.m` och kan bygga på funktionen som skrevs i deluppgift (1b). Bestäm lösningen med $v_0 = 0.9$ fram till tiden $t = 5$. Använd steglängden $h = 0.05$. Programmet ska sedan plotta kulans x-, y- och z-koordinat som funktion av tiden t i samma figur (tre kurvor).

Notera att den framräknade lösningen ges i cylinderkoordinater. Kulans position i kartesiska koordinater ges via sambanden

$$x = r\cos\theta, \qquad y = r\sin\theta, \qquad z = r/\tan\alpha. \tag{5}$$

Uppdatering 19/2-2020

För att få en bild av hur kulan rör sig i konen kan ni med hjälp av funktionen `plotkula.m` som finns i Canvas rita upp kulans bana i 3D. Funktionen anropas som `plotkula(x,y,z)` där `x`, `y` och `z` är kulans x-, y- och z-koordinater som ni beräknat ovan. Prova även att animera kulan med anropet `plotkula(x,y,z,p)` där argumentet `p` styr hur snabb animeringen är. Notera att kulan inte riktigt beter sig som man skulle tro. Detta beror på att friktionen är satt till 0.

b) Beräkna kulans bana med `ode45`

Bygg ut programmet så att det också beräknar kulans bana med MATLABs inbyggda funktion `ode45` för samma fall som i deluppgift (a). Programmet ska sedan plotta banans projektion på xy-planet (dvs x-vektorn plottas mot y-vektorn) för denna lösning och för er Runge-Kutta-lösning, i samma figur (två kurvor).

- Blir det någon synlig skillnad mellan de två lösningarna?

Skicka in: `kulan.m` som innehåller programmet som gör deluppgifterna (a) och (b).

c) Beräkna kulans rörelsemängdsmoment och energi

Kulans rörelsemängdsmoment L [N·m·s] och dess totala energi E [N·m] ges av uttrycken

$$L = mr^2\theta', \qquad E = \frac{mv^2}{2} + mgz. \tag{6}$$

I mekaniken visas att båda dessa storheter i det aktuella fallet bevaras i tiden, dvs de är konstanta. Det är viktigt att även den numeriska lösningen bevarar den här typen av *invarianter* väl för att kunna ge fysikaliskt korrekta förutsägelser. Det kan vara svårt när simuleringstiden är lång och ofta behöver man använda speciella numeriska metoder i det fallet.

Ni ska nu undersöka hur bra metoderna ovan är på att bevara L och E. Förläng simuleringstiden till $t = 125$ och skriv ett MATLAB-program som beräknar L och E på tre sätt:

1. Runge–Kutta 4 med $h = 0.05$,
2. Runge–Kutta 4 med $h = 0.01$,
3. `ode45`.

Programmet ska sedan plotta L som funktion av $t \in [0, 125]$ för alla tre fallen i samma figur (tre kurvor) och E som funktion av t för alla tre fallen i en annan figur (tre kurvor). Använd `legend`-kommandot för att indikera vilken kurva som är vilken. Notera att från en numerisk lösning kan ni beräkna L och E med hjälp av (6) och uttrycken för v och z i (4) respektive (5).

- Vilken metod gör bäst ifrån sig här? Tag hänsyn både till noggrannhet (hur konstant är E och L) och beräkningskostnad (antal tidssteg[1] metoderna använt).

[1] Notera att `ode45` internt gör fyra gånger färre tidssteg än antalet värden som kommandot returnerar. Tre av fyra returnerade värden är framräknade med interpolation.

- Hur bra tror ni Framåt Euler skulle vara för det här problemet?
- (Frivillig uppgift.) Ungefär hur många varv har kulan rört sig runt konen under simuleringstiden 125 sekunder?

Skicka in: `kulan2.m` som innehåller programmet.

d) Beräkna kulans rörelsemängdsmoment och energi nu med friktion

Lägg nu på lite friktion genom att ändra $\mu = 0$ till $\mu = 0.05$ i `fkula.m`. Gå igenom experimenten i deluppgifterna (a)–(c) igen. Hur ändras resultaten? Det är lämpligt att ändra simuleringstiden till till $t = 10$ i (d). (Ni behöver inte skicka in något om detta, men var beredda att redogöra för era slutsatser på redovisningen.)

3. Randvärdesproblem

Följande differentialekvationsproblem beskriver temperaturfördelningen $T(x)$ i en stav av längden L [m].

$$-\frac{d}{dx}\left(k\frac{dT}{dx}\right) = Q(x), \qquad T(0) = t_0, \quad T(L) = t_1. \tag{7}$$

Vänster och höger ändpunkt på staven har den konstanta temperaturen t_0 resp t_1 [K]. Konstanten k [N/(K· s)] är stavens värmeledningsförmåga och $Q(x)$ [N/(m^2·s)] är den värmemängd som per tidsenhet och volymsenhet genereras i staven, tex genom radioaktivitet eller yttre uppvärmning. Antag att $L = 3$, $k = 2$, $t_0 = 290$, $t_1 = 400$ och $Q(x)$ är funktionen

$$Q(x) = q_0\, e^{-q_1(x-0.7L)^2} + 200, \qquad q_0 = 3000, \quad q_1 = 200.$$

Differentialekvation och randvillkor (7) kan lösas numeriskt med hjälp av finita differensmetoden. Diskretisera intervallet $[0, L]$ enligt $x_i = ih$, $i = 0, 1, 2, \ldots, n, n+1$, där $h(n+1) = L$ och låt $T_i \approx T(x_i)$. Efter en approximation av andraderivatan med centraldifferens erhålles

$$\frac{-T_{i-1} + 2T_i - T_{i+1}}{h^2} = \frac{1}{k}Q(x_i), \qquad i = 1, 2, \ldots, n. \tag{8}$$

Tillsammans med randvillkoren $T_0 = t_0$ och $T_{n+1} = t_1$ leder diskretiseringen till ett linjärt ekvationssystem

$$AT = b,$$

där A är en $n \times n$-matris, T är en n-vektor med temperaturvärdena i det inre av intervallet och b är en n-vektor som beror på randvärdena t_0, t_1 och $Q(x_i)$-värdena.

a) Konstruktion av matrisen A och högerled b

Skriv ett MATLAB-program som konstruerar matrisen A och högerledet b för en given storlek n. Matrisen kommer endast att ha ett fåtal nollskilda element. Matrisen skapar du tex med hjälp av MATLABs kommando `diag(v)`, som genererar en diagonalmatris med vektorn `v` på diagonalen, samt generaliseringen `diag(v,p)` som genererar en matris med vektor `v` på diagonal nummer `p`, där `p` kan vara både positiv och negativ.

Tips: Skriv först ner matrisen A med papper och penna för ett enkelt fall, tex $n = 4$, för att se strukturen hos matrisen. Verifiera också att programmet genererar korrekt matris för detta fall.

b) Beräkna temperaturen

Räkna ut temperaturen T genom att lösa det linjära ekvationssystemet $AT = b$ med backslash, \. Pröva först med $n = 10$ inre punkter. Lösningsvektorn T kommer att innehålla temperaturen i alla punkter x_i *förutom* i randpunkterna $x_0 = 0$ och $x_{n+1} = L$. Lägg till dessa randvärden genom att bygga ut T-vektorn med två element. Skapa motsvarande x-vektor och plotta temperaturen som funktion av x på *hela* intervallet $0 \leq x \leq L$.

När detta fungerar ska ni skriva om MATLAB-programmet till en MATLAB-funktion, `stav` som tar `n`, `q0` och `q1` som indata, och returnerar två variabler:

- kolumnvektorn `x` som innehåller alla x-punkter, dvs `x = [0,h,2h, ..., L]`^{T}
- kolumnvektorn `T` som innehåller motsvarande temperaturvärden, dvs `T =` $[T_0, T_1, \ldots, T_n, T_{n+1}]^T$

Notera: Vektorerna `x` och `T` har båda $n+2$ element. Efter ett anrop till `stav.m` ska lösningsplotten ges med kommandot `plot(x,T)`. (Plotkommandot ska dock inte ligga inne i filen `stav.m`!)

Skicka in: Funktionsfilen `stav.m`; anropas som `"[x,T] = stav(n,q0,q1)"`.

c) Tillförlitlighetsanalys

Skriv ett MATLAB-program som beräknar temperaturen för $n = 40, 80, 160, 320$ med hjälp av `stav`-funktionen (med $q_0 = 3000$ och $q_1 = 200$). Programmet ska dels plotta de erhållna resultaten i samma figur (fyra kurvor), dels skriva ut den minimala och maximala temperaturen, samt medeltemperaturen för varje n. Formatera utskriften som en tabell med fyra rader ($n = 40, 80, 160, 320$) och fyra kolumner (n, min T, max T och medel T).

- Hur tillförlitliga är de beräknade värdena? Uppskatta felet i den numeriska lösningen med ~~$N = 80$~~ $N = 320$. (Läs om tillförlitlighetsbedömning i det sista avsnittet i anteckningarna om noggrannhetsordning.)

d) Störningsanalys

Programmet ska också beräkna osäkerheten i den maximala temperaturen när osäkerheten i q_0 och q_1 är given av $q_0 = 3000 \pm 200$ och $q_1 = 200 \pm 10$. Använd experimentell störningsräkning med ett par anrop till `stav`-funktionen. Skriv ut maximala temperaturen och dess osäkerhet. Välj ett n för vilket de numeriska felen är betydligt mindre än osäkerheterna. (Läs om experimentell störningsräkning i anteckningarna om felanalys.)

Skicka in: Filen `temperatur.m` som innehåller programmet som gör deluppgifterna (c) och (d).

4. Sammanställning av filer som ska skickas in

Uppgift 1: `feuler.m`, `rk4.m`, `kvottabell`

Uppgift 2: `kulan.m`, `kulan2.m`

Uppgift 3: `stav.m`, `temperatur.m`