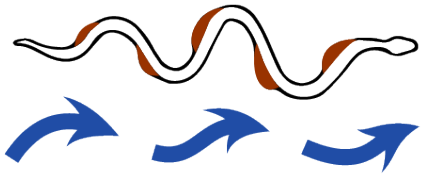
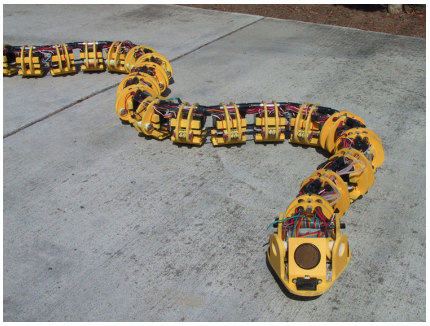
Miniprojekt 3 : Snake locomotion - hur rör sig ormen?

Miniprojektet redovisas först muntligt på övning, och en skriftlig rapport skall inlämnas snart därefter. För datum se kurshemsidan.

Den skriftliga rapporten kan ge max 10 poäng, 5 poäng behövs för godkänt. Dessa poäng är jämnt fördelade på uppgifterna U1-U5. Dessutom finns en bonusuppgift (U6) som ger upp till 2 poäng. Om man gör denna, så kan man uppnå maxpoäng på rapporten även om man fått poängavdrag på de övriga uppgifterna.

På marken rör sig ormar genom att dra ihop och slappna av kroppens längdmuskler på ett koordinerat sätt. Ormarna har på buken utvecklat avlånga fjäll som kallas bukfjäll för att få fäste på underlaget. Då olika delar av kroppen pressar djuret till vänster och andra delar till höger är den resulterande riktningen rakt fram. Det finns arter som kan uppnå en hastighet upp till 6 kilometer per timme.





Figur 1: Ormars rörelsemönster studeras för att förstå optimala rörelsemönster. Ormrobotar (bilden t.h) programmeras för att efterlikna riktiga ormar.

Genom att fästa markörer på en orm, så har koordinaterna för dessa punkter kunna bestämmas vid olika tidpunkter när ormen rör sig. I filen `snakecoord.mat`, som finns på kurshemsidan, är variablerna `xsnake` och `ysnake` lagrade. Dessa är matriser av storlek 501×6, där varje rad beskriver läget i x-led respektive y-led vid en viss tidpunkt för 6 markörer jämnt fördelade längs ormens längd. Antalet rader motsvarar alltså totala antalet tidpunkter som vi registrerat ormens läge för.

För att åskådliggöra ormens rörelse kan vi göra en animation med hjälp av MATLAB-kommandona `plot`, `getframe` och `movie`. Nedanstående kod ger ett prov på hur man kan gå till väga

```
load snakecoord;
%%xsnake and ysnake of sizes noT x (N+1).
noT=size(xsnake,1);
clf;
xs=min(min(xsnake))-0.1;
xe=max(max(xsnake))+0.1;
ys=min(min(ysnake))-0.1;
ye=max(max(ysnake))+0.1;
clear MV;
```

```
for ti=1:noT
    hold off;
    plot(xsnake(ti,:),ysnake(ti,:),'k');
    axis('image');
    axis([xs xe ys ye]);
    axis('off');
    MV(ti)=getframe;
end;
movie(MV);
```

Vid körning av koden kommer ni att märka att ormen får ett kantigt utseende. Detta beror på att vi har få datapunkter per tidssteg och att MATLAB endast förbinder datapunkterna med räta linjer. I första delen av projektet ska ni studera hur man kan förbättra utseendet genom att använda styckvis kubisk interpolation (s k kubiska splines).

Kubisk spline interpolation

U1. Läs kapitel 3.4 i Sauer (s. 166–176) om spline interpolation.

a) En kubisk spline är styckvis kubisk interpolation. Hur ”skarvas” polynomen på de olika delintervallen? Vilka villkor ska vara uppfyllda? Vad gäller i de båda ändarna på hela intervallet? Ge exempel på villkor som kan användas.

Splineinterpolation utförs av Matlabs funktion `spline`, eller helt ekvivalent med `interp1` med valet `'spline'`, som ni använt på demolabben. Använd `help` för att ta reda på vilka ändvillkor som används för funktionen som default. Går det att använda andra ändvillkor?

För att bekanta oss med Matlabs spline funktion så ska vi använda den för en given funktion först, innan vi använder den för mätdata för ormen.

Låt $g(s) = e^{s^2}$ på intervallet $[0, 1]$. Vi väljer ett N och definierar $s_i = (i-1)/N$, $i = 1, \ldots, N+1$, och evaluerar $g(s)$ i dessa punkter. Dvs (med N definierat),

```
s=(0:N)/N;
g=exp(s.^2);
```

Detta är den information som nu skall användas för att definiera en splineapproximation som vi kallar $G^N(s)$. Vi kan definiera en vektor `se` med tätare s punkter och evaluera vår spline funktion i dessa punkter, och spara dessa värden i en vektor `ge`.

b) Låt $N = 10$.

- **i)** Plotta de datapunkter som använts för att definiera splineapproximationen med cirklar. I samma plot så plottar du också splineapproximationen $G^N(s)$ och funktionen $g(s)$. Plotta $G^N(s)$ och $g(s)$ med lika många punkter, och nog många för att få en mjuk kurva.
- **ii)** Plotta skillnaden mellan splineapproximationen och den exakta funktionen. Notera att skillnaden skall bli noll för de s-värden som använts för att definiera splineapproximationen.

b) Bestäm felet $e^N = \|G^N(s) - g(s)\|_\infty$ på intervallet $s \in [0,1]$ för $G^N(s)$ definierad med $N =$ 10, 20, 40 och 80 punkter. För att beräkna felet bör du evaluera splineapproximationen i fler punkter (säg 800 punkter över hela intervallet) och jämföra med originalfunktionen $g(s)$ evaluerad i dessa punkter, och sedan ta max-felet över alla punkter. Plotta felet som en funktion av N med logaritmisk skala på båda axlarna. (Se kommandot `loglog`). Hur beror felet av N?

Förbättring av visuell presentation

Även om vi har glesa data över ormens rörelse så kan vi förbättra den visuella presentationen. Det är förstås naturligt att tänka sig att ormens centerlinje skall vara en mjuk kurva, och inte en kantig en. Om vi utnyttjar en splineapproximation så kan vi konstruera en representation med två kontinuerliga derivator. Med detta så kan vi förbättra filmen över ormens rörelse avsevärt. Baserat på data i `xsnake` och `ysnake` från filen `snakecoord.mat` så kan i varje tidpunkt en splineapproximation för ormens form beräknas, och plottas med fler punkter så att vi får en mjuk kurva.

Här tänker vi att $s = l/L$ är den relativa positionen längs ormens centerlinje, där l är positionen längs linjen, och L är ormens totala längd. Detta betyder att s har värdet 0 i ena ändan av ormen och 1 i andra. Då kan vi betrakta x och y koordinaterna som två olika funktioner av s.

Givet $s_i = (i-1)/N$, $i = 1, \ldots, N+1$ (som anger var längs ormen markörerna sitter) har koordinaterna (x_i, y_i) uppmätts vid olika tidpunkter. För de data som finns i `xsnake` och `ysnake` så är $N = 5$. Vid vald tidpunkt beräknar vi en spline approximation för $x(s)$ och en annan för $y(s)$. När vi evaluerat spline approximationerna i ett antal punkter per intervall och definierat både `xe` och `ye` i analogi med `ge` i U1, så kan vi helt enkelt göra `plot(xe,ye)` för att plotta splineapproximationen av ormens form.

U2. För fyra olika tidpunkter (lite utspridda i tiden), beräkna splineapproximationer för x och y koordinaterna baserat på data från `xsnake` och `ysnake`. Plotta den spline approximation av kurvan som du beräknat och markera i plotten de diskreta punkter du utgått ifrån.

Modifiera nu också koden för animeringen så att splineapproximationen plottas istället för data direkt från `xsnake` och `ysnake`. Ser filmen bättre ut? (Filmen kan visas under muntlig presentation. För skriftlig rapport räcker det med koden för animeringen och plottarna som efterfrågas ovan).

En modell för ormens rörelse

Om krökningen eller *kurvaturen* längs centerlinjen vid en tidpunkt t ges av $\kappa(s,t)$, $s \in [0,1]$, så kan denna integreras för att ge vinkeln som tangentvektorn för kurvan $(x(s,t), y(s,t))$ bildar med x-axeln. Vi kallar denna vinkel för $\theta(s,t)$ och har

$$\theta(s,t) = \theta_0(t) + \int_0^s \kappa(\alpha, t) d\alpha$$

där $\theta_0(t)$ är vinkeln i ändpunkten (svansen) av ormen. När vi har definierat denna vinkel så kan x och y koordinaterna beräknas genom ytterligare en integration

$$x(s,t) = x_0(t) + \int_0^s \cos(\theta(\alpha,t))d\alpha \tag{1}$$

$$y(s,t) = y_0(t) + \int_0^s \sin(\theta(\alpha,t))d\alpha, \tag{2}$$

där (x_0, y_0) är koordinaten för ändpunkten (i svansändan) på ormen och $s \in [0,1]$. Notera att variablen t tiden, ej påverkar integrationen.

Låt $s_i = (i-1)\Delta s$, med $\Delta s = \frac{1}{N}$ och $i = 1, \ldots, N+1$. Vi har att

$$\theta(s_{i+1},t) = \theta(s_i,t) + \int_{s_i}^{s_{i+1}} \kappa(\alpha,t)d\alpha, \quad i = 1,\ldots,N,$$

där $\theta(s_1,t) = \theta(0,t) = \theta_0(t)$. När vi approximerar integralen med trapetsregeln för ett intervall så får vi

$$\tilde{\theta}(s_{i+1},t) = \tilde{\theta}(s_i,t) + \frac{\Delta s}{2}\left(\kappa(s_i,t) + \kappa(s_{i+1},t)\right), \quad i = 1,\ldots,N. \tag{3}$$

där $\tilde{\theta}(s_i,t)$ är en approximation till $\theta(s_i,t)$, och $\tilde{\theta}(s_1,t) = \theta_0(t)$.

Antag nu att kurvaturen är given som

$$\kappa(s,t) = A\cos(k\pi s + 2\pi t),$$

där A och k är givna konstanter. För detta uttryck på $\kappa(s,t)$ så kan vi integrera analytisk och få

$$\theta(s,t) = \theta_0(t) + \frac{A}{k\pi}\left[\sin(k\pi s + 2\pi t)) - \sin(2\pi t)\right]. \tag{4}$$

Att sedan vidare integrera enligt (1) och (2) för att få $x(s,t)$ och $y(s,t)$ måste vi dock göra numeriskt. Då kan vi använda trapetsregeln på samma sätt som beskrivet för $\theta(s,t)$ ovan.

U3. Definiera $s_i = (i-1)\Delta s$, med $\Delta s = \frac{1}{N}$ och $i = 1,\ldots,N+1$. Låt $t = 0$ och $\theta_0(0) = 0$, och sätt $A = 5$ och $k = 3$.

a) Beräkna approximationen $\theta_i^N = \tilde{\theta}(s_i,0)$, för $i = 1,\ldots,N+1$ (för tex $N = 100$) med hjälp av trapetsregeln (3) och jämför med den exakta lösningen för $\theta(s,t)$ (4) som fås genom analytisk integrering av $\kappa(s,t)$. Plotta skillnaden som en funktion av s.

Lite Matlab tips. Med N definierat, gör tex

```
sv=(0:N)/N;
A=5; k=3;
t=0;
thv_ana = (A/(k*pi))*((sin(k*pi*sv + 2*pi*t)) - sin(2*pi*t));
```

Nu vill du definiera en vektor (kalla den tex `thv_num`) vars värden du beräknar med hjälp av trapetsregeln enligt ovan. Du kan jämföra med MATLABs inbyggda funktion `cumtrapz` om du är osäker på om du fått rätt resultat. Den kan du använda enligt

```
ds=1/N;
kappav=A*cos(k*pi*sv+2*pi*t);
thv_numM = ds*cumtrapz(kappav);
```

b) För $N = 50, 100, 200, 400$ beräkna RMS felet

$$e^N_{rms} = \sqrt{((e^N_1)^2 + \cdots + (e^N_{N+1})^2)/(N+1)}.$$

där $e^N_i = \theta^N_i - \theta(s_i, 0)$. Här är $\theta(s_i, 0)$ det analytiska värdet på θ för $s = s_i$ och $t = 0$ och ges av (4). Plotta storleken på felet e^N_{rms} mot N i en log-log plot. (Se kommandot `loglog`). Vad noterar du? Beräkna konvergenshastigheten för trapetsmetoden (du får fram ett värde för varje förfiningssteg, i.e för varje dubblering av N). Hur fort skall felet konvergera med Δs enligt teorin? Stämmer det med det du beräknat här?

U4. Låt $t = 0$ och anta $\theta_0(0) = x_0(0) = y_0(0) = 0$, och sätt återigen $A = 5$ och $k = 3$. Givet $\theta(s, 0)$ enligt (4), så vill vi nu beräkna en approximation till $x(s, 0)$ och $y(s, 0)$ med trapetsregeln.

Skriv kod för att beräkna approximationerna $x^N_i = \tilde{x}(s_i, 0)$ och $y^N_i = \tilde{x}(s_i, 0))$, med trapetsregeln för $s_i = (i-1)\Delta s$, med $\Delta s = \frac{1}{N}$ och $i = 1, \ldots, N+1$. Låt $N = 100$ och plotta ormen i en figur.

Tips: Har du gjort rätt? Ormen ska då börja i punkten $(0, 0)$ och du bör se hela ormen om du gör `axis([-0.05 1.0 -0.05 0.15])` (och sedan `axis('image')` för att skala axlarna proportionellt). Längden på ormen ska vara 1.

U5. Ladda ner filen `snakeendpoint.mat` från kurshemsidan och läs in den i MATLAB med `load(snakeendpoint)`. Den innehåller matrisen `tthxy` som innehåller fyra kolumner. De fyra elementen på varje rad är tiden t_l, $\theta_0(t_l)$, $x_0(t_l)$, $y_0(t_l)$ - varje rad ger data vid en ny tidpunkt. Filen innehåller också A och k som behövs för att definiera $\kappa(s, t)$.

Välj fyra tidpunkter, utspridda i tiden, och beräkna $x(s, t)$ och $y(s, t)$ med din egen kod för trapetsmetoden från U4. Du kan evaluera $\theta(s, t)$ enligt (4), men notera att du nu för tidpunkten t_l behöver lägga till $\theta_0(t_l)$. Glöm inte heller att lägga till $x_0(t_l)$, $y_0(t_l)$ när du beräknar x och y-koordinaterna.

Plotta ormen i en figur för de tidpunkter du valt och ange vilket N du har använt för dina beräkningar. Gör även en animation av ormen utifrån denna modell då du använder all tillgänglig data (dvs alla tidpunkter). (Filmen kan visas under muntlig presentation. För skriftlig rapport räcker det med koden för animeringen och plottarna för de fyra tidpunkterna).

U6. (Extra uppgift). Nu vill vi studera felet och beräkna konvergenshastigheten för vår uträkning av $x(s, t)$ och $y(s, t)$. I det här fallet har vi ingen analytisk lösning för $x(s, t)$ och $y(s, t)$ att jämföra med. Vi kan dock beräkna skillnaden mellan konsekutiva lösningar (lösningar för olika $\Delta s = 1/N$), och ur detta uppskatta konvergenshastigheten. Detta är vad som normalt sett görs, då en analytisk lösning väldigt sällan finns tillgänglig.

Även här låter vi $N = 50, 100, 200, 400$, vilket vi kan skriva som $N_k = 2^{(k-1)}N_0$, med $N_0 = 50$, $k = 1, 2, 3, 4$. I analogi med U4, så kan du kalla x^N_i, approximationen till $x(s_i, \bar{t})$, för vald tidpunkt $\bar{t}$, och motsvarande för y^N_i. För varje N, räkna ut x^N_i och y^N_i, $i = 1, \ldots, N+1$, och spara i en vektor med $N0$ jämnt utspridda punkter, enligt nedan:

```
noN=4;
N0=50;
Nvec=N0*2.^(0:noN-1);
xsave=zeros(noN,N0+1);
ysave=zeros(noN,N0+1);

for l=1:noN
    N=Nvec(l);
    %%==========
    %%Beräkna koordinaterna och spara dem i xvec och yvec.
    %%xvec och yvec radvektorer
    %%==========
    xsave(l,:)=xvec(1:2^(l-1):end);
    ysave(l,:)=yvec(1:2^(l-1):end);
end;
```

I koden ovan behöver du lägga in beräkningen av xvec och yvec som du gjort i U4, detta är bara ett förslag för hur du kan spara resultatet. Nu har du data i `xsave` och `ysave`, som är av storlek `noN x N0+1`.

Vi har nu lösningen i $s_i = (i-1)\Delta s_0 = (i-1)/N0$, $i = 1, \ldots, N0+1$, beräknad med $N = 50, 100, 200, 400$. Tag nu skillnaden mellan lösningar beräknat med $N = 50$ och 100, $N = 100$ och 200, etc och beräkna normen av skillnaden i varje fall. Med vilken faktor minskar det för varje steg? Beräkna konvergenshastigheten.