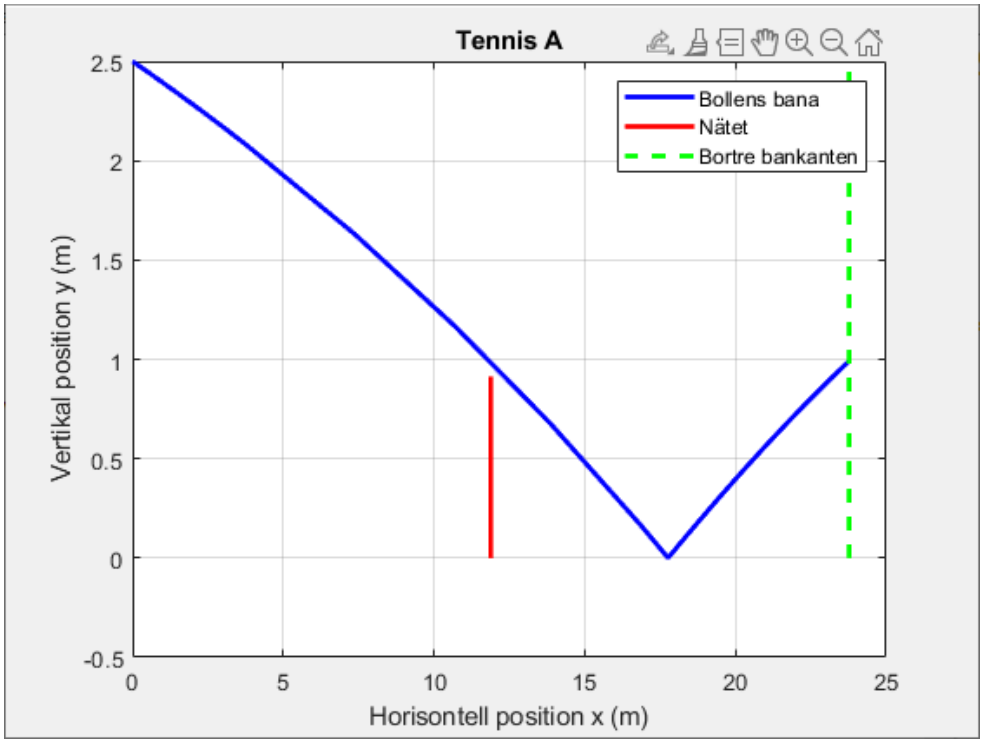
KUNGLIGA TEKNISKA HÖGSKOLAN
Civilingenjörsutbildning i Maskinteknik

Tennis Avancerad

Teeshk Nader & Emil Feratovic



Rapport
SF1512 HT24 Numeriska metoder, grundkurs
Stockholm, 9 december 2024

Sammanfattning

Denna rapport analyserar en tennisserve genom simulering och numeriska metoder för att avgöra om en tennisserve är godkänd, var bollen studsar, höjden vid bortre kant och optimal startvinkel. På basnivån användes Runge-Kutta och sekantmetoden, medan MATLAB:s inbyggda funktioner som **ode45** och **fzero** användes för avancerad analys. Resultaten visade att serven passerade nätet med en höjd på **0,988** meter, första studsen inträffade vid **17,76** meter, och höjden vid bortre kanten var **0,992** meter. En optimal startvinkel på **-5,944°** beräknades för att bollen skulle passera nätet på exakt 1 meters höjd över marken. Jämförelsen av metoder visade att MATLAB:s verktyg förenklade och förbättrade resultatens noggrannhet, men egen implementerade metoder presterade också väl.

Summary

This report analyzes a tennis serve using simulations and numerical methods to determine whether a serve is valid, where the ball bounces, the height at the far edge, and the optimal launch angle. At a basic level, the Runge-Kutta and secant methods were employed, while MATLAB's built-in functions such as **`ode45`** and **`fzero`** were used for advanced analysis. The results showed that the serve cleared the net at a height of **0.988** meters, the first bounce occurred at **17.76** meters, and the height at the far edge was **0.992** meters. An optimal launch angle of **-5.944°** was calculated to ensure the ball passed over the net exactly 1 meter above the ground. The comparison of methods demonstrated that MATLAB's tools simplified and enhanced the accuracy of results, although self-implemented methods also performed well.

Innehållsförteckning

1. Inledning

Syfte

Syftet med detta projekt är att simulera en tennis serve och besvara olika frågeställningar. Dessa kan sammanfattas i 4 olika frågor: blev serven godkänd, given en viss vinkel. Var på banan studsar bollen? Bestämma den optimala vinkeln för att bollen ska passera över marken på exakt en meters höjd och vilken höjd över marken har bollen vid bortre bankanten och vidare.

Metod och tillvägagångssätt

För att besvara dessa frågor används två olika metoder:

- Basnivå: Implementering av numeriska metoder med Runge-Kutta av fjärde ordningen och sekantmetoden utan användning av MATLABs inbyggda funktioner.
- Avancerad nivå: Användning av MATLABs inbyggda funktioner, exempelvis ode45 för lösning av differentialekvationer.

2. Metod

2.1 Basnivå

För att analysera tennisbollens rörelse och besvara deluppgifterna a), b) och c) på basnivå, låg fokus på att lösa de ordinära differentialekvationer som beskriver bollens rörelse under inverkan av gravitation och luftmotstånd, samt att bestämma kritiska punkter i bollens bana.

2.1.1 Modellering av bollens rörelse

Bollens rörelse beskrivs av differentialekvationerna, där krafterna som verkar på bollen inkluderar gravitation och luftmotstånd som ges av:

★ $m\ddot{x} = \ -K\dot{x}V$
★ $m\ddot{y} = \ -mg - K\dot{y}V$
★ $V = \sqrt{\dot{x}^2 + \dot{y}^2}$

Sedan infördes parametrar för att kunna skriva om den till en linjär diffekvation:

$$\begin{cases} u_1 = x \\ u_2 = \dot{x} \\ u_3 = y \\ u_4 = \dot{y} \end{cases}$$

Initialvillkoren sätts baserat på givna startparametrar:

- ★ Startposition: $x_0 = 0\,m,\ y_0 = 2.5$ m
- ★ Startvinkel: θ = -6∘(nedåt mot marken),
- ★ Starthastighet: $\frac{200}{3.6}$m/s (konverterat från km/h till m/s).

Hastigheterna beräknades som:

- ★ $v_{x0} = v_0 cos(\theta),\ v_{y0} = v_0 sin(\theta)$

Tidssteget för simuleringen valdes till Δt = 0.01 s för att säkerställa tillräcklig noggrannhet. Antalet tidssteg N bestämdes utifrån den maximala simuleringstiden $t_{max} = 5\ s$ som testades.

Runge–Kuttametoden tillämpades enligt följande iterativa schema för varje tidssteg:

- $k_1 = f(t_n, y_n)$
- $k_2 = f(t_n + \frac{\Delta t}{2},\ y_n +)$
- $k_3 = f(t_n + \frac{\Delta t}{2},\ y_n + \frac{\Delta t}{2} k_2)$
- $k_4 = f(t_n + \Delta t,\ y_n + \Delta t k_3)$
- $y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

Där y_n är vektorn av tillståndsvariabler vid tidpunkten t_n

2.1.3 Hantering av händelser och begränsningar

För att modellera bollens interaktion med marken och banans gränser implementerades följande:

- **Studs mot marken:** När bollens höjd y blev mindre än eller lika med noll, beräknades positionen för markkontakt genom linjär interpolation mellan de två senaste punkterna. Bollens vertikala hastighet inverterades och multipliceras med en studskoefficient e = 1 (elastisk studs), och simuleringen fortsatte från detta nya tillstånd.
- **Bortre bankanten:** Om bollens horisontella position x översteg 2L (där L = 11,885 m är avståndet till nätet), avslutades simuleringen.
- **Höjden över nätet:** Nätet hade en höjd på 0.914 meter och höjden vid nätets position (x = L) beräknades genom spline-interpolation av de simulerade datapunkterna. Detta möjliggjorde en noggrann bestämning av vilken höjd den hade över nätet samt hur bollen passerade över nätet utan att vidröra det.

2.1.4 Beräkning av höjden vid bortre bankanten

För att besvara deluppgift b) interpolerades bollens höjd vid x = 2L (bortre bankanten) genom linjär interpolation av de simulerade positionerna. Om bollen nådde denna position inom simuleringstiden, kunde höjden direkt beräknas; annars konstaterades att bollen inte nådde bortre bankanten.

2.1.5 Användning av sekantmetoden för optimering av vinkel

I deluppgift c) efterfrågades den startvinkel som resulterar i att bollen passerar över nätet på exakt en meters höjd över marken. För att finna denna vinkel användes sekantmetoden, en numerisk lösningsmetod som inte kräver beräkning av derivator.
Sekantmetoden implementeras enligt följande:

1. **Initiala gissningar:** Två startvärden för vinkeln θ_0 och θ_1 valdes nära den förväntade lösningen. Från lösningen i b) kunde det konstateras att vinkeln definitivt låg mellan -5 och -6 grader.
2. **Iterativ process:** För varje iteration beräknades funktionsvärdet f(θ), definierat som skillnaden mellan den beräknade höjden vid nätet och den önskade höjden (1 meter över marken). Nästa approximation av vinkeln erhölls genom:
   - $\theta_{n+1} = \theta_n - f(\theta)\frac{\theta_n - \theta_{n-1}}{f(\theta_n) - f(\theta_{n-1})}$
3. **Kontroll:** Processen fortsatte tills skillnaden mellan successiva vinklar var mindre än en fördefinierad toleransnivå (10^{-12}) eller maximalt antal iterationer uppnåtts.

Vid varje iteration simulerades bollens bana med den aktuella vinkeln genom Runge-Kutta-metoden, och höjden vid nätet beräknades via spline-interpolation.

2.1.6 Interpolation och grafisk representation

För att visualisera bollens bana och exakt bestämma kritiska höjder användes spline-interpolation på de diskreta datapunkter som genererades av simuleringen. Detta möjliggjorde en smidig kurva som representerar bollens bana. Den grafiska representationen inkluderade:

- Bollens bana i x- och y-led.
- Nätet vid x = L med korrekt höjd.
- Markering av banans planhalvor och bortre bankanten.
- Indikationer på var bollen studsar och dess höjd vid kritiska punkter.

3. Avancerad nivå

För att förbättra effektiviteten och noggrannheten i simuleringen av tennisbollens bana, samt för att undersöka fördelarna med att använda etablerade numeriska metoder, implementerades den avancerade nivån av projektet genom användning av MATLABs inbyggda funktioner. Detta inkluderade utnyttjandet av ode45 för lösning av differentialekvationer och fzero i kombination med ode45 för att optimera startvinkeln.

3.1 Användning av MATLABs inbyggda funktion ode45

3.1.1 Modellering av bollens rörelse med ode45

ode45 är en MATLAB-funktion som löser ordinära differentialekvationer (ODE) med hjälp av en explicit Runge-Kutta-metod av fjärde och femte ordningen.
Rörelseekvationerna för bollen formulerades på samma sätt som i basnivån, men ställdes om för att passa in i syntaxen för **ode45**. Systemet av första ordningens differentialekvationer uttrycktes som:

- $\frac{dy}{dt} = f(t, y)$

där tillståndsvektorn y defineras och högerledet f(t,y) gavs av: $y = \begin{bmatrix} x \\ v_x \\ y \\ v_y \end{bmatrix}$ $f(t, y) = \begin{bmatrix} v_x \\ -\frac{K}{m} v_x v \\ v_y \\ -g - \frac{K}{m} v_y v \end{bmatrix}$

Med : $v = \sqrt{v_x^2 + v_y^2}$

3.1.2 Implementering av ode45

Initialvillkoren sattes enligt:

- Startposition: $x_0 = 0$ m, $y_0 = 2.5$ m.
- Startvinkel: θ (varierande beroende på deluppgift),
- Starthastighet: $v_0 = \frac{200}{3.6} m/s$

Hastighetskomponenterna beräknades som:

- $v_{x0} = v_0 cos(\theta),\ v_{y0} = v_0 sin(\theta)$

För att hantera händelser såsom bollens kontakt med marken eller nå bortre bankanten, definierades en händelsefunktion (**Events**) som övervakar tillståndet under integrationen. När ett fördefinierat villkor uppfylls, exempelvis y = 0 (markkontakt), kan integrationen stoppas eller anpassas.Anropet till ode45 gjordes enligt:

```
options = odeset('Events', @events, 'RelTol', 1e-8, 'AbsTol', 1e-10);
[tid, positioner] = ode45(@tennis_ode, t_span, init_tillstand, options);
```

där:

- @tennis_ode är en funktion som beräknar högerledet f(t,y).
- t_span är tidsintervallet för integrationen.
- init_tillstand är vektorn med initialvillkor.
- options innehåller inställningar för händelser och toleranser.

3.1.3 Händelsehantering

Händelsefunktionen events definierades för att upptäcka när bollen når marken (y = 0) eller bortre bankanten (x = 2L). Funktionen returnerar tre värden:

- **value**: En vektor vars nollställen indikerar att en händelse har inträffat,
- **isterminal**: En vektor som anger om integrationen ska stoppas vid respektive händelse,
- **direction**: En vektor som anger i vilken riktning nollställena ska upptäckas (ökande eller minskande).

För markkontakt och bortre bankanten formulerades händelsevillkoren som:

```
function [value, isterminal, direction] = events(~, y)
  max_x_position = 23.77; % Bortre bankant
  value = [y(3); max_x_position - y(1)]; % [y-position; avstånd till bortre bankant]
  isterminal = [1; 1]; % Avsluta integration vid båda händelserna
  direction = [-1; -1]; % y går mot marken, x går mot bortre kanten
end
```

3.1.4 Hantering av studs och fortsatt integration

Vid markkontakt inverterades den vertikala hastigheten och multipliceras med en studskoefficient **e**, precis som i basnivån. För att fortsätta simuleringen efter en studs, uppdaterades initialvillkoren och **ode45** anropades på nytt med det nya tillståndet och en justerad tidsintervall. En loop användes för att hantera flera potentiella studsar, där varje iteration av loopen representerade en ny fas av bollens rörelse mellan händelser.

3.1.5 Beräkning av höjden vid nätet och bortre bankanten

Efter varje integration användes spline-interpolation för att beräkna höjden vid specifika horisontella positioner, såsom nätets position x=L och bortre bankanten x=2L. Detta gjordes genom:

```
hojd_vid_nat = spline(positioner(:,1), positioner(:,3), nat_x); % Interpolera höjd vid nätet

hojd_vid_bankant = spline(unikt_x, unikt_y, max_x_position);
```

Om bollen inte hade nått dessa positioner inom simuleringstiden, rapporterades att bollen inte nådde så långt.

3.2 Optimering av startvinkel med fzero och ode45

För att bestämma den startvinkel som resulterar i att bollen passerar nätet på exakt en meters höjd över nätet, kombinerades fzero med ode45-baserad simulering.

3.2.1 Implementering av fzero

Fzero implementerades genom att definiera en funktion som, för en given vinkel θ, simulerar bollens bana med ode45:

```
funktion = @(vinkel) berakna_hojd_vid_nat(vinkel, v_start, y_start, K, massa, grav, L) - y_mal;
```

- `vinkel = fzero(funktion, [-6, -5]);`

där "**funktion**" är diffekvationens lösning till y-position 1 meter över nätet

3.2.2 Användning av ode45 inom fzero

Vid varje iteration av ***fzero*** anropas en anonym funktion som simulerar bollens bana med den aktuella vinkeln:

```
function y_net = berakna_hojd_vid_nat(vinkel_deg, ...)
  [y_net, ~, ~] = simulera_bollbana(vinkel_deg, ...);
end
function [y_net, tid, positioner] = simulera_bollbana(vinkel_deg, ...)
  % Konvertera vinkel till radianer
  % Sätt upp initialvillkor
  % Anropa ode45
  % Interpolera höjden vid nätet
end
```

Denna struktur möjliggjorde en modulär och effektiv implementering, där ode45 hanterade integrationen och fzero fokuserade på optimeringen.

3.3 Fördelar med användning av inbyggda funktioner

Användningen av MATLABs inbyggda funktioner erbjöd flera fördelar:

- **Effektivitet**: ode45 är optimerad för prestanda och anpassar automatiskt steglängden för att möta de angivna toleranserna, vilket ofta resulterar i färre funktionsutvärderingar jämfört med fasta steglängdsmetoder.

- **Noggrannhet**: Den adaptiva steglängd anpassningen förbättrar noggrannheten i lösningen, särskilt vid branta förändringar i lösningens derivata.
- **Enkelhet i kod**: Koden blir mer kompakt och lättläst, vilket underlättar underhåll och vidareutveckling.
- **Flexibilitet**: Händelsehantering i ode45 möjliggör enkel implementering av komplexa villkor, såsom flera händelser och anpassade stoppkriterier. Fzero kräver bättre gissningar vilket innebär att intervallet av gissningarna som behöver väljas har större krav av att vara "bra" gissningar vilket kan göra den mindre flexibel än sekant i denna rapports fall.

4. Resultat

4.1 Basnivå

4.1.1 Deluppgift a)

Simuleringen med Runge-Kutta-metoden visade att bollen, vid en starthöjd på 2,50 meter, en startvinkel på -6° och en starthastighet på 200 km/h, följde en bana som kunde interpoleras med hjälp av spline-metoder för att bestämma kritiska punkter, se bild 1. Analysen av bollens höjd vid nätets position (x = 11,885) var höjden 0.98821 meter och tillräcklig för att bollen skulle räknas som en godkänd serve. Vidare kunde studspunkten bestämmas genom linjär interpolation mellan de tidssteg där bollens höjd växlade tecken och nådde marknivån (y = 0). Resultaten visade tydligt var på banan den första studsen inträffade och bollen studsade vid x = 17.7641 meter.

bild 1

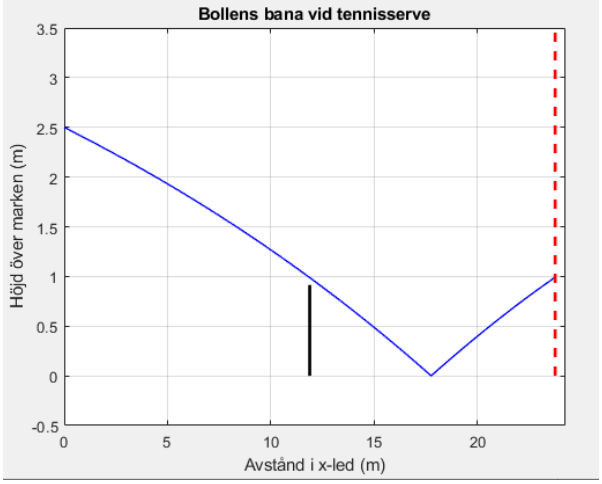
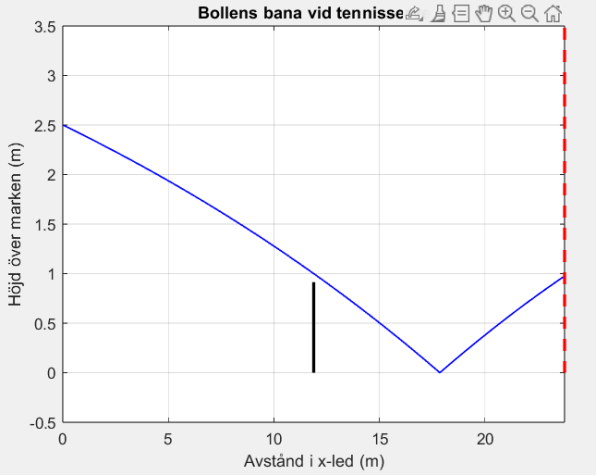


bild 2



4.1.2 Deluppgift b)

För samma initiala villkor som i deluppgift a) beräknades höjden vid banans bortre bankant (x = 23.77 m). Genom användning av spline-interpolation på de beräknade datapunkterna för bollens bana erhölls höjden vid den bortre bankanten. Resultatet visade att höjden vid bortre bankanten är 0.99176 meter. Se bild 2

4.1.3 Deluppgift c)

Vid sökandet efter den vinkel som ger en exakt en meters höjd över nätet användes sekantmetoden i kombination med Runge-Kutta-simuleringen. De initiala gissningarna för vinkeln valdes i närheten av tidigare erhållna värden. Efter ett antal iterationer uppnåddes en vinkel där differensen mellan den uppmätta höjden vid nätet och det önskade värdet (1 meter över marken) var mindre än den angivna toleransen (10^{-12}). Den funna vinkeln resulterade i vinkeln -5.944 grader efter 4 iterationer. Detta gav en y positionen 1.0026 m

4.1.4 Deluppgift d)

Vid en parametervariation på ±1% för starthöjd, starthastighet och startvinkel antogs det att störningsräkning var en lämplig lösning för att ta reda på osäkerheten i de givna parametervariationerna. Störningsräkningen utgick i att störa en parameter i taget med den givna parametervariationen så att kvoten mellan svaret utan störning och med störning blev så stor som möjligt. Detta utfördes genom att störa svaret åt båda hållen och sedan ta kvoten mellan vardera för sig med det faktiska svaret. Den som var större i belopp mellan störningen och det faktiska svaret fick bli täljaren och sedan kunde de jämföras med varandra för att avgöra om en positiv procentuell ökning eller negativ gav störst störning i parametern.

Tabell 1

	Starthöjd +-1%	Starthastighet +-1%	Startvinkel +-1%	Total osäkerhet
a1) x-längd	17.7735-17.9345= -0.1611	17.7735-17.8445= -0.0711	17.7735-17.6907= 0.0827	+-0.3149
a2) y-höjd	0.9882-1.0132= -0.0250	0.9882-0.9934= -0.0052	0.9882-0.9756= 0.0126	+-0.0428
b) y-höjd	0.9892-0.9661= 0.0231	0.9892-0.9735= 0.0157	0.9892-1.0013= -0.0121	+-0.0509
c) vinkel	-5.9441-(-6.0627)= 0.1186	-5.9441-(-5.9686)= 0.0246	-5.9441 -(-6.0035)= 0.0594	+-0.2026

- För x-positionen vid studsen på marken: 17.7641+-0.314868667 ≈ **17.76+-0.3149[m]a)**
- För höjden vid nätet: 0.988210+-0.04278118711 ≈ **0.9882+-0.0428[m]a)**
- För höjden över marken vid bortre bankant: 0.991762+-0.050921≈ **0.9918+-0.0509[m]b)**
- För vinkeln: -5.944056+-0.2025732 ≈ **-5.9441+-0.2026 [grader]c)**

4.2 Avancerad Nivå

bild 3

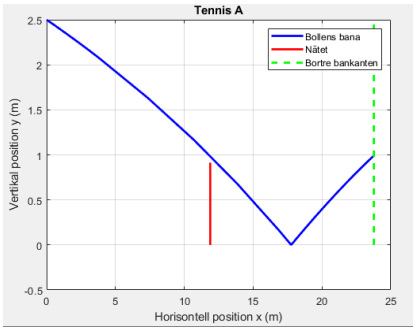
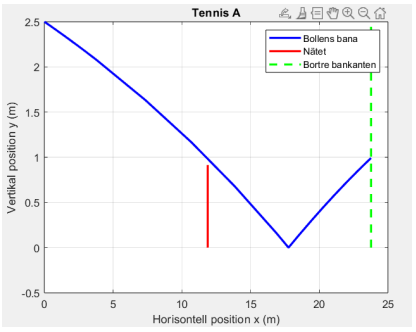


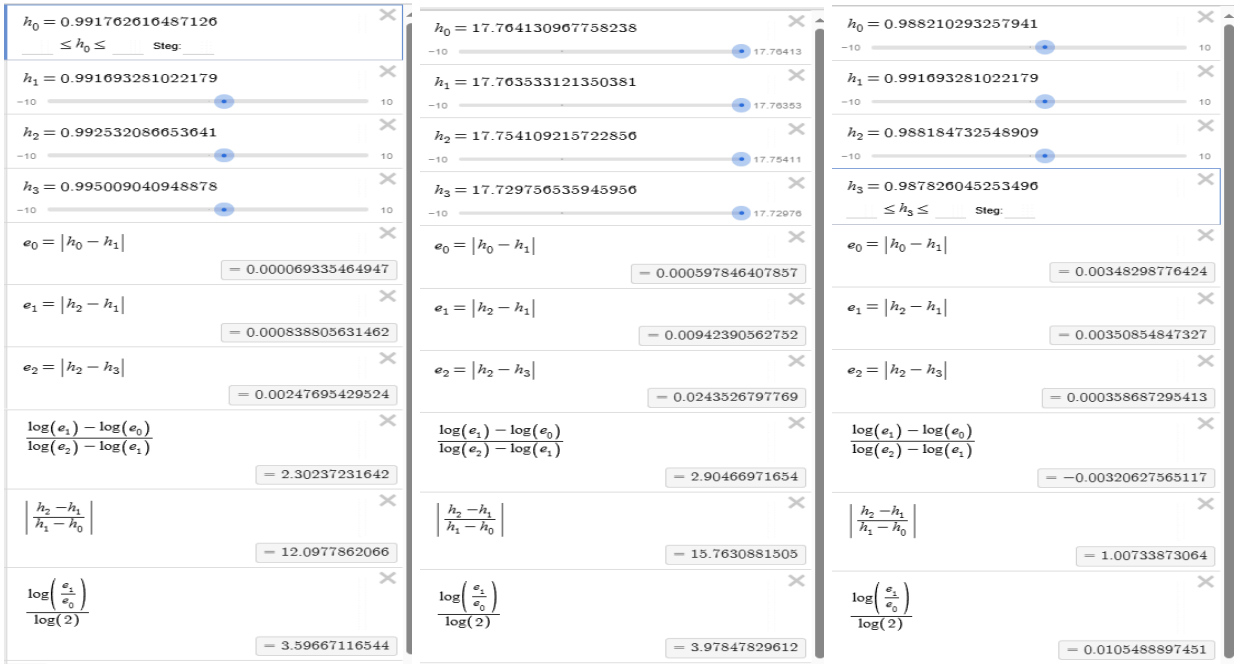
bild 4



4.2.2 Resultatens Överensstämmelse

Resultaten från den avancerade nivån visade i stort sett samma fysikaliska beteende som vid basnivån. Bollens passage över nätet, första studs samt höjden vid bortre bankanten uppvisade endast marginella skillnader, vilka huvudsakligen kan tillskrivas numeriska avrundningsfel och skillnader i metodernas interna felkontroll. De något mer exakta resultaten från den avancerade metoden, i kombination med minskad kodkomplexitet, indikerar fördelarna med att utnyttja väl beprövade inbyggda funktioner.

5. Konvergens och Felanalys



Bankanten **Studsen** **Höjden vid nät**

För att verifiera Runge-Kutta-metodens teoretiska konvergensordning, vilken för en fjärde ordningens metod förväntas vara närmare 4, undersöktes hur felet minskar när tidssteget halveras. I denna analys beräknades fel vid specifika punkter (studspositionen, höjden vid bortre bankanten och höjden vid nätet) för olika tidssteg, varefter konvergensordningen skattades genom att jämföra felen mellan iterationerna. Resultaten visade att för studspositionen uppmättes en konvergensordning på cirka 3,9784, vilket ligger mycket nära det förväntade värdet 4. För höjden vid bortre bankanten erhölls en ordning om 3,5966, vilket fortfarande indikerar god överensstämmelse med den teoretiska ordningen, om än något lägre. Däremot var resultaten för höjden över nätet avvikande, med en observerad konvergensordning på endast cirka 0,01054, eller en felkvot närmare 1. Detta avvikande resultat beror troligen på användandet av splines för att få ut punkten över nätet.

5.1 Feluppskattning

5.1.1 Deluppgift a-b

Basnivå

```
Bollen passerar över nätet utan att nudda det.0.98821meter.
Bollen studsar vid x = 17.7641 meter.
Höjden vid bortre bankanten (x = 23.77 m) är 0.99176 meter.
Bollen studsar vid x = 17.7641 meter.
(Feluppskattning) Studs x-linjär = 17.7641, x-kvadratisk = 17.7641, diff = 3.5527e-15
(Feluppskattning) Bankant y-linjär = 0.99176, y-kvadratisk = 0.99176, diff = 9.992e-16
(Feluppskattning) Nätet y-linjär = 0.98818, y-kvadratisk = 0.98821, diff = 2.8944e-05
```

En feluppskattning av interpolationen genom att jämföra en linjär metod med en kvadratisk visade i båda fallen ytterst små skillnader, mellan 10^{-15} och 10^{-16} meter för studs- och bankantpositionerna. Vid nätets position uppmättes en avvikelse på ~ $3 \cdot 10^{-5}$ meter mellan linjär och kvadratisk interpolation, vilket fortfarande är försumligt i sammanhanget. Dessa resultat indikerar att interpolationsfelet är **mycket litet** för de kritiska händelserna i deluppgift a–b.

Avancerad

```
a) Bollen går över nätet utan att nudda det. Höjd vid nätet: 0.9882 meter.
(Feluppskattning) Kvadratisk vs Spline vid nätet: 0.00029733 m
Bollen studsar vid x = 17.7646 meter.
b) Höjden vid bortre bankanten (x = 23.77 m) är 0.99181 meter.
(Feluppskattning) Kvadratisk vs Spline vid bankant: 3.5527e-15 m
```

En jämförelse mellan kvadratisk interpolation och spline vid nätets x-position indikerade en liten differens på ungefär $0.00029733m$, vilket tyder på att interpolationsfelet är förhållandevis lågt just där. För samma simulering vid bortre bankanten uppmättes en skillnad på endast $3.55 \cdot 10^{-15}m$, vilket är försumbart i detta sammanhang.

Dessa resultat visar att användningen av ode45 i kombination med spline ger en noggrann modell av bollens bana och höjd vid kritiska punkter. Den minimala avvikelsen mellan kvadratisk interpolation och spline påvisar att interpolationsfelet är mycket litet, i synnerhet vid bortre bankanten.

5.1.2 Deluppgift c

Basnivå

```
1. Spline vs. Quad vid nätet: 1.65e-07 m
2. Spline vs. Quad vid nätet: 2.7968e-07 m
3. Spline vs. Quad vid nätet: 2.7968e-07 m
4. Spline vs. Quad vid nätet: 1.72e-07 m
5. Spline vs. Quad vid nätet: 1.72e-07 m
6. Spline vs. Quad vid nätet: 1.7198e-07 m
7. Spline vs. Quad vid nätet: 1.7198e-07 m
8. Spline vs. Quad vid nätet: 1.7198e-07 m
Den sökta vinkeln är -5.9441
```

Avancerad

```
1. Spline vs. Quad vid nätet: 4.4173e-07 m
2. Spline vs. Quad vid nätet: 3.5812e-07 m
3. Spline vs. Quad vid nätet: 4.3645e-07 m
4. Spline vs. Quad vid nätet: 4.3646e-07 m
5. Spline vs. Quad vid nätet: 4.3646e-07 m
6. Spline vs. Quad vid nätet: 4.3646e-07 m
Den sökta vinkeln är -5.9441 grader.
```

Vid varje iteration jämfördes spline- och kvadrat interpolations värden för bollens höjd vid $x = L$. I **basnivån** uppmättes differenser kring $1.65 \cdot 10^{-7}m$ till $2.8 \cdot 10^{-7}m$för de första iterationerna, och därefter stabiliserade sig felet runt $1.72 \cdot 10^{-7}m$ i de sista iterationerna. Detta indikerar att interpolationsfelet är mycket litet när den sökta vinkeln närmar sig konvergens.

I **avancerad nivå**, där ode45 och inbyggda funktioner utnyttjades, uppmättes liknande små differenser mellan spline och kvadratisk interpolation, från omkring $4.42 \cdot 10^{-7} m$ ned till $3.58 \cdot 10^{-7} m$, och stabiliserade sig slutligen nära $4.36 \cdot 10^{-7} m$. De minimala avvikelserna i båda fallen bekräftar att interpolationsfelet är försumligt i sammanhanget och att den sökta vinkeln runt −5,944 i samtliga tester kan bestämmas med hög noggrannhet oavsett om man använder egen Runge-Kutta-lösning (basnivå) eller ode45 (avancerad nivå).

6. Diskussion

Jämförelsen mellan de två angreppssätten, basnivå med egenimplementerade numeriska metoder och avancerad nivå med MATLABs inbyggda funktioner visar att det i huvudsak erhålls liknande resultat. På basnivå, där Runge-Kutta-metoden av fjärde ordningen och sekantmetoden implementerades manuellt, kunde bollens bana, nätpassage och höjd vid bortre bankanten bestämmas med god noggrannhet. Metoderna krävde dock noga valda tidssteg och toleranser för att säkerställa stabilitet och korrekt konvergens. Konvergensanalysen indikerade att Runge-Kutta-metoden i stort sett uppnådde förväntad ordning runt 4, men även att vissa mätpunkter inte fullt ut återspeglade den teoretiska konvergensordningen. Detta understryker komplexiteten hos problemet och betydelsen av att noggrant välja tidssteg samt att analysera resultaten för olika delmängder av problemet.

7. Slutsats

Att simulera en tennisserve, bestämma om serven är godkänd, hitta var bollen studsar första gången, beräkna höjden vid bortre bankanten och optimera startvinkeln för att uppnå en specifik höjd över nätet. Har uppnåtts genom både egenimplementerade numeriska metoder och användning av MATLABs inbyggda funktioner. Resultaten från basnivån visar att Runge-Kutta-metoden av fjärde ordningen och sekantmetoden kan ge tillräckligt noggranna lösningar för att besvara samtliga deluppgifter, om än med viss manuell anpassning av tidssteg och toleranser. Genom att tillämpa avancerade inbyggda metoder kunde arbetet förenklas och exaktheten höjas med mindre kodansträngning. Markkontakt och bortre bankantdetektion hanterades elegant med inbyggda funktioner som ode45 och fzero.

8. Referenser

1. “ODE Event Location”. U.d. Mathworks, ODE Event Location, Hämtad: [2024-11-25]

9. Bilagor

Runge kutta a-b bas nivå

```
clear all; close all; clc;
format long
% Konstanter
massa = 0.057;      % Massa i kg
grav = 9.82;        % Tyngdacceleration
K = 0.001;          % Luftmotstånd
L = 11.885;         % Avstånd till nätet i meter
v_start = 200 / 3.6; % Startfart i m/s
vinkel = -6;        % (nedåt)
y_start = 2.5;      % Start från 2.5 m höjd
% Tidssteg och total tid
dt = 0.01*1;         % Tidssteg i sekunder
t_max = 5 ;          % Maximal simuleringstid i sekunder
N = floor(t_max / dt);    % Antal tidssteg
% Initiala hastighetskomponenter
hastighet_x0 = v_start * cosd(vinkel);  % Hastighet i x-led
hastighet_y0 = v_start * sind(vinkel);  % Hastighet i y-led
% Initialt tillstånd
U = zeros(4, N);   % [x; x'; y; y']
U(:,1) = [0; hastighet_x0; y_start; hastighet_y0];
% Funktion för derivatorna
V = @(U) sqrt(U(2)^2+U(4)^2);
uprim = @(U) [U(2); -K * U(2) * V(U) / massa; U(4); -grav - K * U(4) * V(U) /
massa];
% Studskoefficient
e = 1;  % Ändra till ett värde mellan 0 och 1 för energiförlust
% Runge-Kutta loop
for i = 1:N-1
 % Beräkna k1, k2, k3, k4 för Runge-Kutta
 k1 = uprim(U(:,i));
 k2 = uprim(U(:,i) + dt/2 * k1);
 k3 = uprim(U(:,i) + dt/2 * k2);
```

```
 k4 = uprim(U(:,i) + dt * k3);
 k = (k1 + 2 * k2 + 2 * k3 + k4) / 6;
 % Uppdatera positioner och hastigheter
 U(:,i+1) = U(:,i) + dt * k;
% Hantera studs mot marken
 if U(3,i+1) <= 0
     % Interpolera för att hitta exakt tidpunkt för markkontakt
     alfa = U(3,i) / (U(3,i) - U(3,i+1));
     % Uppdatera tillståndet vid markkontakten
     U(:,i+1) = U(:,i) + alfa * (U(:,i+1) - U(:,i));
     U(3,i+1) = 0;  % Sätt höjden till 0
     U(4,i+1) = -e * U(4,i+1);  % Invertera och skala den vertikala hastigheten
     % Kontrollera om den vertikala hastigheten är för liten för att fortsätta
     if abs(U(4,i+1)) < 1e-5
         U = U(:,1:i+1);
         break;
     end
 end
 % Hantera bortre bankanten
 if U(1,i+1) >= 2 * L
     % Interpolation vid bortre bankanten
     beta = (2 * L - U(1,i)) / (U(1,i+1) - U(1,i));
     U(:,i+1) = U(:,i) + beta * (U(:,i+1) - U(:,i));
     U(1,i+1) = 2 * L;  % Sätt x-positionen till bortre bankanten
     U = U(:,1:i+1);
     break;
 end
end
% Kontrollera om bollen passerar över nätet utan att nudda det
nat_x = L;       % Nätets position i x-led
nat_y = 0.914;   % Nätets höjd i meter
% Spline för att hitta höjden vid nätet
y_vid_nat = spline(U(1,:), U(3,:), nat_x);
if y_vid_nat > nat_y
 disp(['Bollen passerar över nätet utan att nudda det.', num2str(y_vid_nat),
'meter.']);
else
 disp(['Bollen traffar nätet (x = ', num2str(nat_x), ' m) vid ',
num2str(y_vid_nat), ' meter.']);
end
% Plotta bollbanan med datapunkter
figure;
plot(U(1,:), U(3,:), 'b-', 'LineWidth', 1);
hold on;
% Rita nätet
nat_x = L;       % Nätets position i x-led
nat_y = 0.914;   % Nätets höjd i meter
```

```
plot([nat_x, nat_x], [0, nat_y], 'k', 'LineWidth', 2);
% Plotta bollbanan med splines
xx = linspace(min(U(1,:)), max(U(1,:)), 1000);
yy = spline(U(1,:), U(3,:), xx);
% Markera serverutan
serveruta_start = 11.885;
serveruta_slut = 11.885 + 11.885;
plot([serveruta_slut, serveruta_slut], [0, max(yy)+1], 'r--', 'LineWidth', 2);
xlabel('Avstånd i x-led (m)');
ylabel('Höjd över marken (m)');
title('Bollens bana vid tennisserve');
grid on;
xlim([0, max(U(1,:))]);
ylim([0, max(U(3,:)) + 1]);
% Hitta var bollen studsar första gången
studsar = find(U(3,:) == 0);
if ~isempty(studsar)
 studs_x = U(1, studsar(1));
 disp(['Bollen studsar vid x = ', num2str(studs_x), ' meter.']);
else
 disp('Ingen studs hittades inom simuleringstiden.');
end
% b)
% Beräkna höjden vid bortre bankanten
bankant_x = 2 * L;  % x = 23.77 meter
% Kontrollera om bollen har nått bortre bankanten
if U(1,end) >= bankant_x
 y_vid_bankant = spline(U(1,:), U(3,:), bankant_x);
 disp(['Höjden vid bortre bankanten (x = ', num2str(bankant_x), ' m) är ',
num2str(y_vid_bankant), ' meter.']);
else
 disp('Bollen har inte nått bortre bankanten inom simuleringstiden.');
end
% ------------------------------------------------
% a) Feluppskattning för studsen
% ------------------------------------------------
studsar = find(U(3,:) == 0);
if ~isempty(studsar)
   studs_index = studsar(1);  % Första studs
   studs_x = U(1, studs_index);
   disp(['Bollen studsar vid x = ', num2str(studs_x), ' meter.']);

   % Om studsen inträffar mellan i och i+1, har vi redan gjort
   % en linjär interpolation i koden (genom "alfa"). Nu vill vi jämföra
   % med en kvadratisk approximation:
   if studs_index > 1 && studs_index < size(U,2)
       % Anropa hjälpfunktionen som ger linjär vs kvadratisk
```

```
       [x_lin, x_quad, err_studs] = estimateStudsX(U, studs_index);
       disp(['(Feluppskattning) Studs x-linjär = ', num2str(x_lin), ...
             ', x-kvadratisk = ', num2str(x_quad), ...
             ', diff = ', num2str(err_studs)]);
   end
else
   disp('Ingen studs hittades inom simuleringstiden.');
end
% ------------------------------------------------
% b) Feluppskattning för bortre bankanten
% ------------------------------------------------
% Hitta index j så att U(1,j) < 2L <= U(1,j+1)
j_candidates = find(U(1,:) < bankant_x & [U(1,2:end),NaN] >= bankant_x);
if ~isempty(j_candidates)
   j = j_candidates(1);
   [y_lin_bank, y_quad_bank, err_bank] = estimateY(U, j, bankant_x);
   disp(['(Feluppskattning) Bankant y-linjär = ', num2str(y_lin_bank), ...
         ', y-kvadratisk = ', num2str(y_quad_bank), ...
         ', diff = ', num2str(err_bank)]);
end
% ------------------------------------------------
% Feluppskattning för nätet (linjär vs. kvadratisk)
% ------------------------------------------------
j_candidates_net = find(U(1,:) < nat_x & [U(1,2:end), NaN] >= nat_x);
if ~isempty(j_candidates_net)
   jn = j_candidates_net(1);
   [y_lin_net, y_quad_net, err_net] = estimateY(U, jn, nat_x);
   disp(['(Feluppskattning) Nätet y-linjär = ', num2str(y_lin_net), ...
         ', y-kvadratisk = ', num2str(y_quad_net), ...
         ', diff = ', num2str(err_net)]);
end
% ------------------------------------------------
% Hjälpfunktioner
% ------------------------------------------------
function [x_lin, x_quad, interp_error] = estimateStudsX(U, studs_idx)
% estimateStudsX - Jämför linjär och kvadratisk interpolation för studs.
%   Använder de punkter U(1,studs_idx-1 : studs_idx+1) och U(3,...)
%   Returnerar x_lin (linjär), x_quad (kvadratisk) och skillnaden err.
   % Indata
   xVals = U(1,:);
   yVals = U(3,:);
   % Linjär interpolation gjordes redan i koden via 'alfa',
   % men här återskapas den för jämförelse.
   i = studs_idx - 1;  % studs sker mellan i och i+1
   if i < 1, i = 1; end
   x0 = xVals(i);    y0 = yVals(i);
   x1 = xVals(i+1);  y1 = yVals(i+1);
```

```
    alfa = y0/(y0 - y1);
    x_lin = x0 + alfa*(x1 - x0);
    % Kvadratisk interpolation runt (i-1, i, i+1) om möjligt
    i_low = max(1, i-1);
    i_high = min(i_low + 2, length(xVals));
    i_low = i_high - 2;  % försök säkra 3 punkter
    if i_low < 1, i_low = 1; end
    XX = xVals(i_low:i_high);
    YY = yVals(i_low:i_high);
    % Polynom av grad 2: y = a x^2 + b x + c
    p = polyfit(XX, YY, 2);
    % Lös p(x) = 0
    r = roots(p);
    % Välj den rot som ligger närmast x_lin
    [~, idxMin] = min(abs(r - x_lin));
    x_quad = r(idxMin);
    % Feluppskattning
    interp_error = abs(x_quad - x_lin);
end
function [y_lin, y_quad, interp_error] = estimateY(U, j, x_pos)
% estimateY - Jämför linjär och kvadratisk interpolation för
%   y-positionen när x = x_bankant. och nätet
    xVals = U(1,:);
    yVals = U(3,:);
    x0 = xVals(j);
    x1 = xVals(j+1);
    y0 = yVals(j);
    y1 = yVals(j+1);
    % Linjär interpolation
    beta = (x_pos - x0) / (x1 - x0);
    y_lin = y0 + beta*(y1 - y0);
    % Kvadratisk interpolation
    j_low = max(1, j-1);
    j_high = min(j_low+2, length(xVals));
    j_low = j_high - 2;
    if j_low < 1, j_low = 1; end
    XX = xVals(j_low:j_high);
    YY = yVals(j_low:j_high);
    p = polyfit(XX, YY, 2);
    y_quad = polyval(p, x_pos);
    interp_error = abs(y_quad - y_lin);
end
```

Runge kutta c uppgift basnivå

```
clear all; close all; clc;
% Konstanter
massa = 0.057;            % Massa i kg
grav = 9.82;              % Tyngdacceleration
K = 0.001;                % Luftmotstånd
L = 11.885;               % Avstånd till nätet i meter
v_start = 200 / 3.6;      % Startfart i m/s
y_start = 2.5;            % Starthöjd i meter
nat_y = 0.914;            % Nätets höjd i meter
y_mal = 1.00;     % Önskad höjd vid nätet
% Funktion för att beräkna höjden vid nätet
funktion = @(vinkel) berakna_hojd_vid_nat(vinkel, v_start, y_start, K, massa, grav,
L) - y_mal;
% Sekantmetodens parametrar
theta0 = -6;              % Första gissning i grader
theta1 = -5;              % Andra gissning i grader
tolerans = 1e-12;         % Tolerans för konvergens
max_iter = 100;           % Max antal iterationer
% Sekantmetoden
for iter = 1:max_iter
 f0 = funktion(theta0);
 f1 = funktion(theta1);
 theta2 = theta1 - f1 * (theta1 - theta0) / (f1 - f0);
 if abs(theta2 - theta1) < tolerans
     vinkel = theta2;
     break;
 end
 theta0 = theta1;
 theta1 = theta2;
end
% Kontrollera om lösning hittades
if iter == max_iter
 disp('Ingen lösning hittades inom max antal iterationer.');
else
 disp(['Den sökta vinkeln är ', num2str(vinkel)]);
end
% ------------------------------------------
% Funktioner
% ------------------------------------------
function y_net = berakna_hojd_vid_nat(vinkel, v_start, y_start, K, massa, grav, L)
 [U, i] = simulera_bollbana(vinkel, v_start, y_start, K, massa, grav, L);
 xdata = U(1,1:i+1);
 ydata = U(3,1:i+1);
```

```
 y_net_spline = spline(xdata, ydata, L);
 % Kvadratisk interpolation
 y_net_quad = Quad(xdata, ydata, L);
 % Feluppskattning
 error_est = abs(y_net_spline - y_net_quad);
 persistent callCount
 if isempty(callCount)
    callCount = 0;
 end
 callCount = callCount + 1;
 disp([num2str(callCount),'. Spline vs. Quad vid nätet: ', num2str(error_est), '
m']);
 % Returnera spline-värdet
 y_net = y_net_spline;
end
function [U, i] = simulera_bollbana(vinkel_deg, v_start, y_start, K, massa, grav,
L)
 % Simulerar bollens bana för en given vinkel och returnerar U och sista index i
 % Konvertera vinkel till radianer
 vinkel_rad = deg2rad(vinkel_deg);
 % Initiala hastighetskomponenter
 hastighet_x0 = v_start * cos(vinkel_rad);
 hastighet_y0 = v_start * sin(vinkel_rad);
 % Tidssteg och total tid
 dt = 0.01;  % Använd ett litet tidssteg för noggrannhet
 t_max = 5;
 N = floor(t_max / dt);
 % Initialt tillstånd
 U = zeros(4, N);   % [x; x'; y; y']
 U(:,1) = [0; hastighet_x0; y_start; hastighet_y0];
 V = @(U) sqrt(U(2)^2+U(4)^2);
 uprim = @(U) [U(2); -K * U(2) * V(U) / massa; U(4); -grav - K * U(4) * V(U) /
massa];
 % Runge-Kutta loop
  for i = 1:N-1
     % Beräkna k1, k2, k3, k4 för Runge-Kutta
     k1 = uprim(U(:,i));
     k2 = uprim(U(:,i) + dt/2 * k1);
     k3 = uprim(U(:,i) + dt/2 * k2);
     k4 = uprim(U(:,i) + dt * k3);

     k = (k1 + 2 * k2 + 2 * k3 + k4) / 6;
     % Uppdatera positioner och hastigheter
     U(:,i+1) = U(:,i) + dt * k;
     % Hantera studs mot marken
     if U(3,i+1) <= 0
         U(4,i+1) = -U(4,i+1);        % Invertera hastigheten i y-led
```

```
    end
    % Avsluta simuleringen om bollen lämnar banan
    if U(1,i+1) >= 2 * L  % Banan slutar vid x = 2 * L
        break;
    end
 end
end
function y_val = Quad(xData, yData, x_pos)
   % Hitta index för närmaste xData kring x_pos
   [~, idxMin] = min(abs(xData - x_pos));
   % Försök plocka ut idxMin-1, idxMin, idxMin+1
   i_low = max(1, idxMin-1);
   i_high = i_low + 2;
   if i_high > length(xData)
       i_high = length(xData);
       i_low = i_high - 2;
       if i_low < 1, i_low=1; end
   end
   % x_lokal, y_lokal
   Xlok = xData(i_low : i_high);
   Ylok = yData(i_low : i_high);
   % Polyfit grad 2
   p = polyfit(Xlok, Ylok, 2);
   % Beräkna y(x_query)
   y_val = polyval(p, x_pos);
end
```

Avancerat nivå a-b

```
clear all; close all; clc;
format long
% ------------------------------------------------
% Konstanter och initiala parametrar
% ------------------------------------------------
% Startparametrar
start_hastighet = 200/3.6; % m/s
start_vinkel = deg2rad(-6); % Startvinkel i radianer
start_x = 0; % Startposition i x-led
start_y = 2.50; % Startposition i y-led
hastighet_x = start_hastighet * cos(start_vinkel); % Initial hastighet i x-led
hastighet_y = start_hastighet * sin(start_vinkel); % Initial hastighet i y-led
% Fysiska konstanter
```

```
massa = 0.057; % kg
luftmotstand = 0.001; % Ns^2/kg^2
gravitation = 9.82; % m/s^2
studs_koefficient = 1; % Elastisk studs
% Gränser och hinder
max_x_position = 23.77; % Bortre bankant
nat_x = 11.885; % Nätets x-position
nat_hojd = 0.914; % Nätets höjd
% Initialtillstånd
init_tillstand = [start_x; hastighet_x; start_y; hastighet_y];
% Simuleringsinställningar
options = odeset('Events', @events);
tid_total = [];
positioner_total = [];
tid_intervall = [0, 5]; % Tid för varje integration
nat_passerad = false;
% ------------------------------------------------
% Simulering
% ------------------------------------------------
while tid_intervall(1) < tid_intervall(2)
 % Lös differentialekvationerna för nuvarande tidsintervall
 [tid, positioner, tid_event, pos_event, id_event] = ode45(@tennis_ode,
tid_intervall, init_tillstand, options);
 % Spara lösningen
 tid_total = [tid_total; tid];
 positioner_total = [positioner_total; positioner];
 % Kontrollera om bollen passerar nätet
 if ~nat_passerad && any(positioner(:,1) >= nat_x)
     hojd_vid_nat = spline(positioner(:,1), positioner(:,3), nat_x); % Interpolera
höjd vid nätet
      % --- Feluppskattning: Kvadratisk vs Spline ---
       [~, quad_nat, err_nat] = Error(positioner(:,1), positioner(:,3), nat_x);
     if hojd_vid_nat > nat_hojd
         disp(['a) Bollen går över nätet utan att nudda det. Höjd vid nätet: ',
num2str(hojd_vid_nat), ' meter.']);
     else
         disp('a) Bollen nuddar nätet.');
     end
     % Visa feluppskattningen i meter
     disp(['(Feluppskattning) Kvadratisk vs Spline vid nätet: ', num2str(err_nat),
' m']);
     nat_passerad = true;
 end
 % Hantera eventuella studsar
 if ~isempty(pos_event)
     % Kontrollera om bollen når bortre bankanten
     if pos_event(1) >= max_x_position
```

```
        break; % Avsluta simuleringen
    end
    % Studspositionen
    disp(['Bollen studsar vid x = ', num2str(pos_event(1)), ' meter.']);

    % Uppdatera tillståndet för en studs
    init_tillstand = [pos_event(1); pos_event(2); 0; -studs_koefficient *
pos_event(4)];
    tid_intervall = [tid_event, tid_intervall(2)]; % Uppdatera tid för nästa
iteration
 else
    break; % Ingen mer händelse, avsluta
 end
end
% ------------------------------------------------
% Analysera resultat
% ------------------------------------------------
% Filtrera unika värden för spline
[unikt_x, unikt_index] = unique(positioner_total(:,1), 'stable');
unikt_y = positioner_total(unikt_index, 3);
% Kontrollera höjden vid bortre bankanten
if max(unikt_x) >= max_x_position
 hojd_vid_bankant = spline(unikt_x, unikt_y, max_x_position);
 disp(['b) Höjden vid bortre bankanten (x = ', num2str(max_x_position), ' m) är ',
num2str(hojd_vid_bankant), ' meter.']);
 % --- Feluppskattning (Kvadratisk vs Spline) ---
 [~, quad_bank, err_bank] = Error(unikt_x, unikt_y, max_x_position);
 disp(['(Feluppskattning) Kvadratisk vs Spline vid bankant: ', num2str(err_bank), '
m']);
else
 disp('b) Bollen når inte bortre bankanten inom simuleringen.');
end
% ------------------------------------------------
% Plotta resultat
% ------------------------------------------------
figure;
plot(unikt_x, unikt_y, 'b-', 'LineWidth', 2); % Bollens bana
hold on;
plot([nat_x, nat_x], [0, nat_hojd], 'r-', 'LineWidth', 2); % Nätet
plot([max_x_position, max_x_position], [0, max(unikt_y)], 'g--', 'LineWidth', 2); %
Bortre bankant
xlabel('Horisontell position x (m)');
ylabel('Vertikal position y (m)');
title('Tennis A');
legend('Bollens bana', 'Nätet', 'Bortre bankanten');
grid on;
% ------------------------------------------------
```

```
% Funktioner
% ------------------------------------------------
% Funktion för rörelseekvationer
function dydt = tennis_ode(~, y)
 m = 0.057;
 K = 0.001;
 g = 9.82;
 x2 = y(2);
 y2 = y(4);
 V = sqrt(x2^2 + y2^2);
 dydt = [x2;
         (-K * x2 * V) / m;
         y2;
         (-g - (K * y2 * V) / m)];
end
% Händelsefunktion för marknivå och bortre bankant
function [value, isterminal, direction] = events(~, y)
 max_x_position = 23.77; % Bortre bankant
 value = [y(3); max_x_position - y(1)]; % [y-position; avstånd till bortre bankant]
 isterminal = [1; 1]; % Avsluta integration vid båda händelserna
 direction = [-1; -1]; % y går mot marken, x går mot bortre kanten
end
% ------------------------------------------------
% Hjälpfunktion för feluppskattning (Kvadratisk vs Spline)
% ------------------------------------------------
function [spline_val, quad_val, error_est] = Error(xData, yData, x_query)
   % Beräkna spline-värdet vid x_query
   spline_val = spline(xData, yData, x_query);
   % Leta upp närmaste index kring x_query
   [~, idxMin] = min(abs(xData - x_query));
   idx_range = (idxMin-1):(idxMin+1);
   idx_range = idx_range(idx_range >= 1 & idx_range <= length(xData));
   % Om vi inte får exakt 3 punkter, välj de 3 närmaste
   if length(idx_range) < 3
       [~, sortIdx] = sort(abs(xData - x_query));
       idx_range = sortIdx(1:3);
   end
   % Ta ut de lokala punkterna
   x_local = xData(idx_range);
   y_local = yData(idx_range);
   % Kvadratisk polynomfit
   p = polyfit(x_local, y_local, 2);
   quad_val = polyval(p, x_query);
   % Feluppskattning
   error_est = abs(spline_val - quad_val);
end
```

Avancerat nivå C

```
clear all; close all; clc;
format long
% ------------------------------------------------
% Konstanter
% ------------------------------------------------
massa = 0.057;      % Massa i kg
grav = 9.82;        % Tyngdacceleration
K = 0.001;          % Luftmotstånd
L = 11.885;         % Avstånd till nätet i meter
v_start = 200 / 3.6;% Startfart i m/s
y_start = 2.5;      % Start från 2.5 m höjd

% Målvärde för höjden vid nätet
y_mal = 1.0; % 1 meter över marken

% ------------------------------------------------
% Funktion för höjd vid nätet minus målvärde
% ------------------------------------------------
funktion = @(vinkel) berakna_hojd_vid_nat(vinkel, v_start, y_start, K, massa, grav,
L) - y_mal;

% Anropa fzero med ett intervall eller en startgissning
vinkel = fzero(funktion, [-6, -5]);

disp(['Den sökta vinkeln är ', num2str(vinkel), ' grader.']);

% ------------------------------------------------
% Funktioner
% ------------------------------------------------
function y_net = berakna_hojd_vid_nat(vinkel_deg, v_start, y_start, K, massa, grav,
L)
   [y_net, ~, ~] = simulera_bollbana(vinkel_deg, v_start, y_start, K, massa, grav,
L);
end

function [y_net, tid, positioner] = simulera_bollbana(vinkel_deg, v_start, y_start,
K, massa, grav, L)
   % Simulerar bollens bana med ode45 och beräknar höjden vid nätet

   % Konvertera vinkel till radianer
```

```
    vinkel_rad = deg2rad(vinkel_deg);

    % Initiala hastighetskomponenter
    hastighet_x0 = v_start * cos(vinkel_rad);
    hastighet_y0 = v_start * sin(vinkel_rad);

    % Initialtillstånd: [x; x'; y; y']
    init_tillstand = [0; hastighet_x0; y_start; hastighet_y0];

    % Tidsintervall och odeset-inställningar
    t_span = [0, 5];
    options = odeset('Events', @events, 'RelTol', 1e-8, 'AbsTol', 1e-10);

    % Kör simuleringen med ode45
    [tid, positioner] = ode45(@tennis_ode, t_span, init_tillstand, options);

    % Extrahera x- och y-positioner
    x_vals = positioner(:, 1); % x-positioner
    y_vals = positioner(:, 3); % y-positioner

    % Kontrollera om bollen har nått nätets position
    if x_vals(end) >= L
        % Använd spline för att interpolera höjden vid nätet
        y_net = spline(x_vals, y_vals, L); % Höjd vid nätets x-position
    else
        % Om bollen inte har nått nätet, använd sista värdet
        y_net = y_vals(end);
    end
end

function dydt = tennis_ode(~, y)
    % Differentialekvationer för bollens rörelse
    m = 0.057; % Massa
    K = 0.001; % Luftmotståndskoefficient
    g = 9.82;  % Gravitation

    % Hastigheter
    vx = y(2); % Hastighet i x-led
    vy = y(4); % Hastighet i y-led
    v_tot = sqrt(vx^2 + vy^2); % Total hastighet

    % Differentialekvationer
    dydt = [vx;
              (-K * vx * v_tot) / m;
              vy;
              (-g - (K * vy * v_tot) / m)];
end
```

```
function [value, isterminal, direction] = events(~, y)
   % Händelsefunktion för att stoppa integrationen när bollen når marken
   value = y(3); % y-position
   isterminal = 1; % Stoppa integrationen vid händelsen
   direction = -1; % Händelse sker när y är på väg ned
end
```