TANA23 Matematiska algoritmer och modeller

Laboration 3. Splines

1 Introduktion

Redovisning sker genom att en kort rapport skrivs där svaret på alla frågor finns med. Program skrivna i Python skall bifogas. Tänk på att programmen skall skickas på ett format som gör att jag kan testköra dem. Enklast är att bara lägga in all kod i en textfil med lite kommentarer angående vilken uppgift som ett visst kodstycke löser. Bifoga även grafer. Den färdiga rapporten skall skickas med epost till `fredrik.berntsson@liu.se`. Var noga med att skriva *TANA23 Lab3* i rubriken.

2 Interpolation med splines

I Python finns en rutin `CubicSpline` som kan användas för att beräkna interpolerande kubiska splines. Vi får tillgång till den genom att skriva

```
>>>from scipy.interpolate import CubicSpline
```

I denna övning skall vi interpolera funktionen

$$f(x) = \frac{4}{3}x^4 - \frac{4}{3}x^3 + \frac{1}{2}x^2,$$

på intervallet $0 \leq x \leq 1$, med en kubisk spline.

Uppgift 2.1 Välj $n = 5$ interpolationspunkter, på intervallet $0 \leq x \leq 1$, med `x=np.linspace(0,1,n)`. Hitta sedan den kubiska splinefunktion $s_h(x)$, med *naturliga ändpunktsvillkor*, som interpolerar $f(x)$ punkterna. Plotta både funktionen $f(x)$ och splinefunktionen $s_h(x)$ på aktuellt intervall. Plotta även det absoluta felet $|f(x) - s_h(x)|$ på samma intervall. Vad kan du säga om felet? **Redovisa graferna**

Eftersom de naturliga ändpunktsvillkoren endast är en approximation så gör vi ett fel vid $x = 0$ och $x = 1$. Detta bör leda till att felet i approximationen blir större nära $x = 0$ och $x = 1$. För att undvika extra stora fel just vid ändpunkterna kan *rätta* ändpunktsvillkor användas.

Uppgift 2.2 Beräkna $f'(0)$ och $f'(1)$.

Uppgift 2.3 Välj återigen $n = 5$ interpolationspunkter på intervallet $0 \leq x \leq 1$ och hitta den kubiska splinefunktion $s_h(x)$, med *rätta ändpunktsvillkor*, som interpolerar $f(x)$ punkterna. Plotta både funktionen $f(x)$ och splinefunktionen $s_h(x)$ på aktuellt intervall. Plotta även det absoluta felet $|f(x) - s(x)|$ på samma intervall. Vad kan du säga om felet nu? **Redovisa graferna**

Från teorin vet vi att trunkeringsfelet vid interpolation med kubiska splines är

$$R_T = |f(x) - s_h(x)| \approx C \cdot h^p,$$

där C är en konstant, h är steglängden, och p är ett heltal. Med n, jämt utspridda, interpolationspunkter på ett intervall $a \leq x \leq b$ fås $n-1$ delintervall och alltså är $h = (b-a)/(n-1)$.

Uppgift 2.4 Interpolera funktionen $f(x)$ ovan med kubiska splinefunktioner. Använd $n = 11$, $n = 21$ och $n = 41$, interpolationspunkter och *rätta ändpunktsvillkor*, dvs $s_h'(0) = f'(0)$ och $s_h'(1) = f'(1)$. Beräkna det maximala felet $|s_h(x) - f(x)|$ för de olika värdena på n. Fyll i och **redovisa tabellen**.

h	1/10	1/20	1/40
$\lvert R_T(h)\rvert$			

Uppgift 2.5 Vi skall nu utnyttja att $R_T(h) \approx C \cdot h^p$ för att hitta p. Utnyttja uttrycket för R_T och visa att

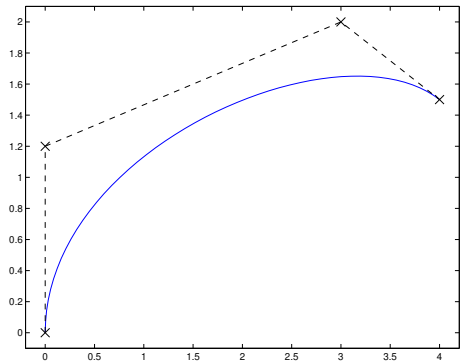
$$\frac{|R_T(h_1)|}{|R_T(h_2)|} \approx \left(\frac{h_1}{h_2}\right)^p.$$

Utnyttja sedan tabellen ovan för att bilda kvoter som borde få värdet 2^p och bestäm därigenom p. Stämmer ditt p med teorin? **Redovisa både beräkningarna och** p

3 Beziér kurvor och design av en font

I många tillämpningar behöver man beskriva ett objekts form matematiskt. Ett exempel är då ett dokument skall skrivas ut. Vi behöver beskriva de enskilda bokstävernas form. Helst skall den matematiska beskrivningen vara sådan att det är enkelt att generera bitmap bilder av bokstäverna i önskad upplösning. Ett fontpaket skapas genom att en konstnär skapar en grafisk representation av varje bokstav. Varje enskild bokstav beskrivs sedan matematiskt genom att bokstavens kontur approximeras med ett givet funktionsuttryck. Proceduren beskrivs i boken *The Metafont book*, av Donald E. Knuth (skaparen av språket TeX). I denna övning skall vi skapa en matematisk beskrivning av bokstaven d. Vi skall även visa hur kursiv text kan skapas från grundbeskrivningen. Standard metoden för att beskriva bokstäver är Beziér kurvor. Oftast kubiska sådana.

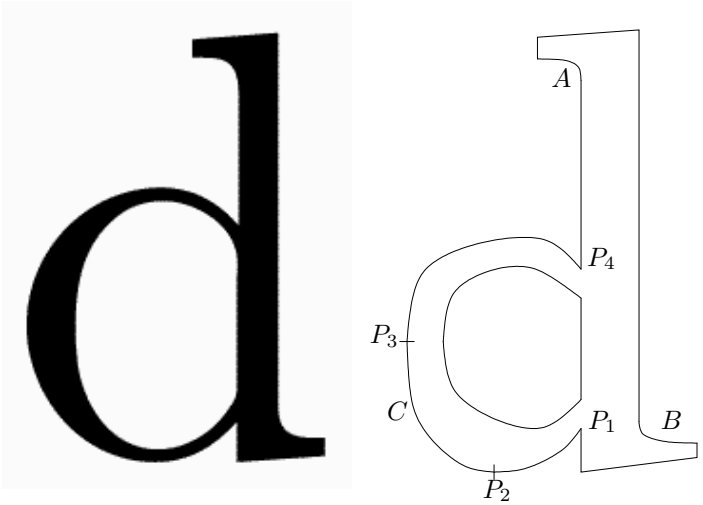
Som start för övningen finns en fil `DrawFont.py` där all kod skall skrivas in. Kodstycken som inte längre behövs kan senare kommenteras bort.



Uppgift 3.1 En kubisk Beziér kurva definieras av 4st *styrpunkter* P_1, P_2, P_3 och P_4. Kurvan ges av uttrycket

$$p(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t) t^2 p_3 + t^3 P_4, \qquad 0 \le t \le 1.$$

Skriv en Python funktion `DrawBezierCurve`, med 4 styrpunkter och ett heltal n som inparametrar. Funktionen skall skapa en vektor `t=np.linspace(0,1,n)` och beräkna Beziér kurvan $p(t)$ för samtliga t-värden. Funktionen skall även plotta både kurvan och styrpunkterna. Försök få det så likt exemplet i bilden ovan som möjligt.



Figur 1: Bokstaven d skapad från en vektoriserad font. Bokstavens form beskrivs av kubiska Beziérkurvor och linjesegment. Några interpolations punkter visas också.

Uppgift 3.2 Programmet `DrawFont` ritar upp de linjesegment som behövs för att rita bokstaven. Flera interpolationspunkter finns lagrade i en matris `Points`. Vi skall nu lägga till de kurvbitar som saknas. Börja med att skapa kurvsegmentet A. Låt P_1 och P_2 vara start, respektive slut-, punkt för kurvstycket. Dessa finns redan i matrisen `Points`. Välj sedan två styrpunkter C_1 och C_2 på ett sådant sätt att kurvan har en vertikal tangent vid startpunkten. Vilket villkor garanterar detta? Vi vill även ha en horisontell tangent i slutpunkten. Hur skall detta garanteras? Lägg till ett anrop till `DrawBezierCurve` så att kurvstycket ritas ut. Blev det snyggt? **Redovisa valet av styrpunkter med motivering**.

Uppgift 3.3 Nu skall vi skapa den kurva som utgör den yttre bågen C. Kurvan skall börja i P_1 och sluta i P_4. Eftersom vi vill att kurvan skall tangera linjen $y = 0$ så väljer vi en extra interpolationspunkt $P_2 = (x_2, 0)$. Du måste experimenteraq lite för att hitta ett lämpligt värde på x_2. Vi vill även att kurvans tangent skall vara vertikal i någon punkt. Detta blir nästa interpolationspunkt P_3.

När interpolationspunkterna är valda så skall vi nu skapa de extra styrpunkter som behövs. Vi har tre kursvstycken $P_1 \mapsto P_2$, $P_2 \mapsto P_3$, och $P_3 \mapsto P_4$. Vi behöver två extra styrpunkter för varje kurvavsnitt. Styrpunkterna skall väljas så att kraven ovan uppfylls. Experimentera lite så att resultatet blir bra.

Tips Det är enklast att lägga till interpolationspunkter i matrisen `Points` och styrpunkter i matrisen `Control`.

Uppgift 3.4 Slutligen skall vi skapa de två återstående kurvavsnitten. Den inre bågen skapas på precis samma sätt som den yttre. Välj två nya interpolations punkter och skapa tre Beziér kurvor. Dessutom behövs en ny Beziér kurva för att skapa den lilla bågen i nedre högra hörnet av bokstaven. **Redovisa bilden med den slutliga bokstaven. Notera att din funktion `DrawBezierCurve` fortfarande skall rita ut styrpunkterna förutom själva kurvan.**

Uppgift 3.5 Alternativa utseenden, som **Tjock** eller *Kursiv*, skapas genom att en matematisk avbildning appliceras på interpolations- och styrpunkter innan bokstaven ritas upp. För att få lutande stil använder vi avbildningen $(x, y) \mapsto (x+\alpha y, y)$. Denna är förberedd i Python programmet. Ett lämpligt värde är $\alpha = 0.07$. Resultatet skall bli en lätt lutande bokstav. Experimentera gärna lite med olika värden på α.

Uppgift 3.6 Nu skall du kommentera bort den del av `DrawBezierCurve` som ritar ut styrpunkterna. Använd sedan `DrawFont` för att rita ut den slutgiltiga versionen av bokstaven d. Blev det bra? **Redovisa bilder på bokstaven både i vanlig och kursiv variant.**

Kommentar Beziér kurvor är ett standardverktyg för kurvdesign. De flesta datorgrafik paket, som `OpenGL` eller `Postscript`, har stöd för solika typer av splinekurvor. Hur fontpaket skapas beskrivs väldigt bra i boken *TeX och METAfont*, av Donald E. Knuth, 1979.