Projekt: Räckvidd för elbil

Du utvecklar en kartapp och färddator för elektriska fordon. Appen ska utökas med funktioner för återstående körsträcka längs en given rutt för aktuell laddning av fordonets batteri.

Bilars energiförbrukning beror till stor del på hastigheten. Räckvidden längs med olika rutter kan därför variera kraftigt. I det här projektet ska ni ta fram en uppskattning av återstående körsträcka (räckvidd) givet information om energiförbrukning vid olika hastigheter samt förväntad hastighet längs med given rutt.

För utvecklingssyfte kommer ni att använda data över energiförbrukningen för den eldrivna sportbilen Tesla Roadster och hastighetsdata längs med en viss rutt som ni tagit fram med hjälp av gps-spårare. När den nya funktionaliteten är klar så ska den kunna användas med data för andra bilmodeller och förväntad hastighet längs med given rutt beräknad utifrån gällande hastighetsbestämmelser, historiska hastighetsdata, och/eller aktuell trafikinformation.

Tesla Roadster 2.5 at AutoRAI Amsterdam 2011 by Overlaet / Wikimedia Commons / CC-BY-SA-3.0

P1, Introduktion (uppvärming)

Ladda ned och packa upp zip-arkivet `roadster.zip`. Arkivet innehåller kod och data att utgå ifrån vid projektarbetet. Viss funktionalitet är redan implementerad. I varje del av projektet arbetar du vidare på koden från föregående del. Din uppgift är att implementera ett antal funktioner vars funktionshuvuden är givna men som saknar implementationer och i nuläget genererar fel vid anrop. I arkivet finns också kod för att testa dessa funktioner. Ett nödvändigt villkor för godkänt resultat är att testerna på respektive del går igenom. I avsnittet *Tester och referens* på sid. 6 finns en beskrivning av hur du kör testerna. Notera att testerna inte täcker all programmering som krävs för att genomföra projektet.

(a) Konsumtionsdata: För att uppskatta bilens räckvidd behöver vi, för given bilmodell, elkonsumtionen som funktion av hastighet. Modellen $c(v) = a_1 v^{-1} + a_2 + a_3 v + a_4 v^2$ har anpassats till förbrukningsdata för Tesla Roadster, där hastigheten v uppmätts i km/h och elkonsumtionen c uppmätts i Wh/km. Detta har gett följande koefficienter:

a_1	a_2	a_3	a_4
546.8	50.31	0.2594	0.008210

Skriv en Python-funktion `consumption` som returnerar elkonsumtionen $c(v)$ (Wh/km) för given hastighet v (km/h). Funktionen ska ha funktionshuvudet

```
def consumption(v):
```

som finns i filen `roadster.py`, men du behöver alltså redigera filen och implementera funktionen.

Man ska kunna anropa funktionen för godtycklig positiv hastighet. Funktionen ska kunna anropas med en NumPy-array och ska då returnera en array med den energiförbrukning som svarar mot varje värde i input-arrayen. Ett sådant anrop finns i skriptet `script_part1a.py`. Utgå ifrån detta skript och redigera det så att det också plottar energiförbrukning som funktion av hastighet i spannet av hastigheter mellan 1 och 200 km/h.

(b) Ruttdata: Dina kollegor Anna och Elsa har kört mellan två städer längs varsin rutt. Med hjälp av gps-spårare har de registrerat position och hastighet. De har sedan bearbetat datat för att få fram hastigheten för ett antal punkter längs med rutten. De har sparat datat för de två rutterna i två filer `speed_anna.npz` och `speed_elsa.npz`.

Du ska nu skriva ett program som läser in data för en rutt och plottar hastighetsdata som funktion av tillryggalagd sträcka. Du kan använda den givna funktionen `load_route` för att läsa in ruttdata lagrad i en `.npz`-fil. Information om denna och andra funktioner fås med kommandot `help`. I detta fall, tex

```
import roadster
help(roadster.load_route)
```

Plotta hastigheten längs med rutten som ett punktdiagram för att illustrera att vi endast har data i ett begränsat antal punkter.

Vi behöver kunna läsa av hastigheter också emellan de givna datapunkterna. Vi tillhandahåller i `roadster.py` en funktion `velocity(x,route)` som returnerar hastighet $v(x)$ (km/h) för godtycklig tillryggalagd sträcka x (km) inom ruttens totala längd. Parametern `route` är namnet på `.npz`-filen där hastighetsdata för rutten ligger lagrad så att funktionen kan anropas tex `v=velocity(x,'speed_anna')`. Funktionen använder interpolation för att anpassa en funktion till givet data.[1]

Uppdatera figuren ovan med ruttdata med en plot av den anpassade funktionen och undersök om den beter sig som förväntat. Tips: Notera att ett stort antal punkter kan behövas för att lösa upp funktionernas kraftiga svängningar. Zooma in i plotten för att bättre se hur funktionen beter sig mellan givna datapunkter.

P2, Integration (area under kurvan)

(a) Ankomsttid: Du ska nu använda funktionen `velocity` från P1(b) för att uppskatta hur lång tid det tar att färdas en viss sträcka längs en viss rutt. Tiden $T(x)$ (h) det tar att färdas x km längs med rutten ges av

$$T(x) = \int_0^x \frac{1}{v(s)} \mathrm{d}s. \tag{1}$$

[1] Interpolation innebär anpassning av en funktion så att den går exakt genom de givna datapunkterna. Funktionen kan användas för approximation i mellanliggande värden. Funktionen `velocity` använder "pchip"-interpolation vilket ger en interpolerande funktion med kontinuerlig förstaderivata som är monoton mellan datapunkterna.

Du kan anta att $v(s)$ är positiv. Formulera trapetsmetoden för att uppskatta integralen och skriv en Python-funktion som returnerar tid $T(x)$ (h) för given tillryggalagd sträcka x (km). Funktionshuvudet för funktionen som du ska implementera finns i filen `roadster.py`:

```
def time_to_destination(x, route, N):
```

Här är `route` som tidigare och N är antalet punkter i trapetsmetoden ($n = N - 1$ delintervall). Notera att du skall göra en egen implemention av trapetsmetoden, utan anrop tex till funktioner för numerisk integration i exempelvis NumPy eller SciPy.

Hur lång tid tar det för Anna och Elsa att färdas hela sträckan längs sina respektive rutter? Undersök hur många punkter N som krävs för att få korrekt antal minuter i den numeriska integrationen.

(b) Total elkonsumtion: Du ska nu använda funktionerna `velocity` och `consumption` från P1 för att uppskatta elkonsumtionen för en given sträcka längs en viss rutt. Den totala elkonsumtionen $E(x)$ i wattimmar efter en tillryggalagd sträcka på x km ges av

$$E(x) = \int_0^x c(v(s))\mathrm{d}s \tag{2}$$

där $c(v)$ är elkonsumtion per km (Wh/km) som funktion av hastighet v (km/h) och $v(s)$ är förväntad hastighet längs med rutten. Formulera som ovan trapetsmetoden för att uppskatta integralen och skriv en Python-funktion som returnerar elkonsumtion $E(x)$ (Wh) för given tillryggalagd sträcka x (km). Funktionen ska ha funktionshuvudet

```
def total_consumption(x, route, N):
```

som finns i filen `roadster.py`, där du ska göra din implementation.

Hur mycket energi gör Anna och Elsa totalt av med längs med sina respektive rutter? Undersök hur många punkter N som krävs för att få korrekt antal wattimmar i den numeriska integrationen.

(c) Konvergensstudie: För åtminstone en av de två integralerna (1) och (2) skall du nu göra en empirisk undersökning av noggrannhetsordningen för din metod.

Räkna ut integrationsfelet för en serie beräkningar där antalet delintervall $n = N - 1$ dubbleras successivt, dvs för en serie $n, 2n, 4n, 8n, 16n, \ldots$ (d.v.s. anropa din funktion med $N = n{+}1, 2n{+}1, 4n{+}1, 16n{+}1, \ldots$). Om du inte känner till det exakta integralvärdet kan du jämföra med integralvärdet med dubbel steglängd. Alltså, för beräkningen med n delintervall jämför du med värdet med $n/2$ delintervall, etc.

Plotta integrationsfelet som funktion av antalet delintervall n eller steglängd $h = (x - 0)/n$. Här är det vanligtvis lämpligt att plotta i log-skala, det kan du göra genom att istället för `plot` i matplotlib.pyplot använda kommandot `loglog` som ger log-skala både på x- och y-axeln. Plotta också hjälplinjer som visar vilken lutning på kurvan som svarar mot olika noggrannhetsordningar. Om du förväntar dig noggrannhetsordning p plottar du en hjälplinje som visar $O(1/n^p)$- eller $O(h^p)$-beteende. Hitta en linje som stämmer överens med dina resultat.

Stämmer den empiriskt bestämda noggrannhetsordningen med teorin för trapetsmetoden? Om inte, varför?

P3, Icke-linjära ekvationer (hitta roten)

(a) Sträcka: Antag nu att föraren vill veta hur långt längs rutten hon förväntas ha kommit på en viss tid. Utgå från ekvationen för ankomsttid och formulera den icke-linjära ekvation vars rot ger förväntad tillryggalagd sträcka efter T timmar. Formulera sedan Newtons metod för att lösa ekvationen. Fundera på vad som skulle vara en bra startgissning för att hitta roten. Skriv en Python-funktion där du implementerar Newtons metod inklusive startgissning. Funktionen ska ha funktionshuvudet

```
def distance(T, route):
```

som finns i filen `roadster.py`. Funktionen ska returnera förväntad tillryggalagd sträcka x (km). Funktionen skall beräkna roten x med ett fel som är mindre än 10^{-4}. Tänk på att felet beror både på noggrannheten i den numeriska integrationen och stoppvillkoret för Newtons metod. Hur långt kommer Anna och Elsa längs sina respektive rutter på 30 min?

Tips: Använd funktionerna `velocity` och `time_to_destination`.

(b) Räckvidd: Antag nu att batteriet är laddat till C Wh. Utgå från ekvationen för total elkonsumtion och formulera den icke-linjära ekvation vars rot ger bilens räckvidd. Formulera sedan Newtons metod för att lösa ekvationen. Fundera på vad som skulle vara en bra startgissning för att hitta roten. Skriv en Python-funktion där du implementerar Newtons metod inklusive startgissning. Funktionen ska ha funktionshuvudet

```
def reach(C, route):
```

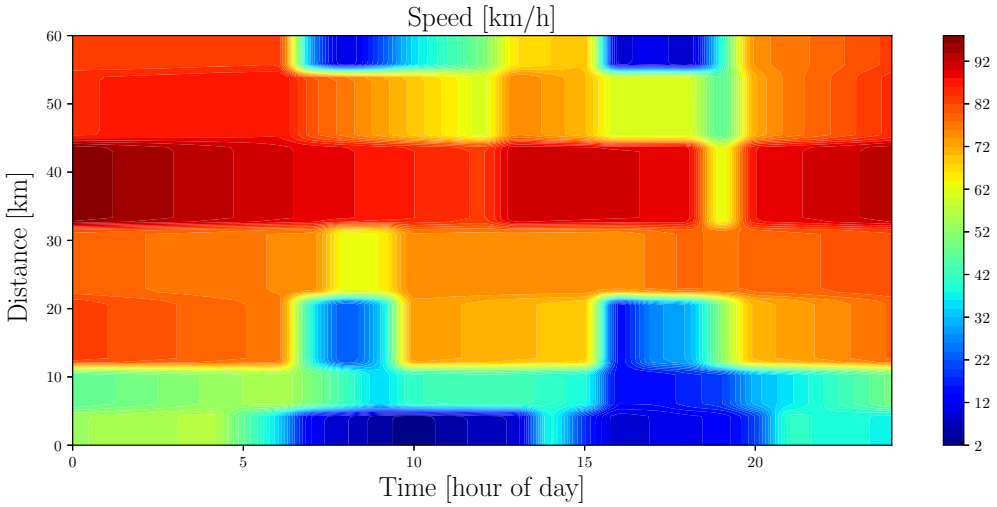
som finns i filen `roadster.py`. Funktionen ska returnera räckvidden x (km). Funktionen skall som ovan beräkna roten med ett fel som är mindre än 10^{-4}. I de fall batteriets laddning är tillräcklig för hela rutten skall ruttens totala längd returneras. Hur långt skulle Anna och Elsa komma på sina respektive rutter med en batteriladdning på $C = 10000$ Wh?

Tips: Använd funktionerna `total_consumption`, `velocity` och `consumption`.

P4, Ordinära differentialekvationer (lös initialvärdesproblemet)

Vi antar nu att hastigheten längs med rutten är given som en funktion av position och tidpunkt under dagen. Denna funktion av två variabler kan vara baserad på historiska hastighetsdata och/eller nyligen uppdaterad trafikinformation. Uppgiften är att givet en sådan funktion och en starttid simulera en färd längs med rutten. Denna simulering ger direkt en uppskattning av ankomsttid givet starttid och kan användas för att generera ruttfiler som input till tidigare funktioner såsom till exempel funktionen `reach` från P3(b).

För att testa koden har vi en modell som bygger på registrerade genomsnittshastigheter för ett antal vägavsnitt i New York under ett antal onsdagar mellan 2017 och 2020. Modellen ger förväntad hastighet givet tidpunkt (h) under dagen och position (km) längs med den 60 km långa rutten, se figuren nedan.



Modellen ges av funktionen `route_nyc(t,x)` i modulen `route_nyc.py`. Funktionen kan anropas tex

```
speed = route_nyc(8.5,20)
```

för att få förväntad hastighet för ett fordon som kl 08.30 befinner sig 20 km in i rutten.

(a) Resa längs rutt En färd längs med rutten med starttid $t_0 \in [0, 24]$ beskrivs av differentialekvationen

$$\begin{cases} x'(t) &= f(t,x), \quad t_0 \le t \le 24,\ 0 \le x \le 60, \\ x(t_0) &= 0, \end{cases} \tag{3}$$

där $f(t,x)$ alltså är implementerad i `route_nyc(t,x)`. Du ska nu skriva en funktion som använder Eulers metod för att lösa differentialekvationen och simulerar en färd längs med rutten med start vid en given tidpunkt t_0 och som avslutas när hela sträckan tillryggalagts, det vill säga när $x = 60$. Funktionen ska ha funktionshuvudet

```
def nyc_route_traveler_euler(t0, h):
```

och finns i filen `route_nyc.py`. Funktionen ska använda Eulers metod med steglängd h. Ett sista Eulersteg med steglängd h skulle i allmänhet resultera i en position $x > 60$. Därför ska det sista steget kortas så att $x = 60$ uppnås exakt i det sista Eulersteget. Du kan anta att fordonet ankommer till målet tidigare än kl 24.00. Funktionen ska kunna anropas

```
time_h, distance_km, speed_kmph = route_nyc_traveler_euler(t0, h)
```

där `time_h` är en NumPy-array med tidpunkter (i timmar efter 0.00) och `distance_km` och `speed_kmph` är NumPy-arrayer med position (km) och hastighet (km/h) vid dessa tidpunkter genererade av Eulers metod som beskrivet ovan. De första värdena i `time_h` kommer att vara t_0, $t_0 + h$, $t_0 + 2h, \ldots$.

Hur lång tid skulle det ta att färdas hela sträckan givet starttid kl 04.00 respektive kl 09.30?

(b) Figur: Python-kod för att plotta figuren ovan över `route_nyc(t,x)` är given i filen `script_part4b.py`. Uppdatera koden så att ruttfärderna med start kl 04.00 respektive kl 09.30 plottas i figuren. Notera att varje färd längs med rutten ges av en kurva som startar vid tiden t_0 och position $x = 0$ och slutar vid $x = 60$.

P5, ej obligatoriska extrauppgifter

(a) Generera ruttfiler: För en given starttid t_0, använd `route_nyc_traveler_euler` för att generera en ruttfil lik `speed_anna.npz` och `speed_elsa.npz`. Använd funktionen `save_route` som finns i `roadster.py`. Kör `time_to_destination` med den nya ruttfilen som input och verifiera att den ger samma tid som `route_nyc_traveler_euler`. Notera hur tidsstegningen med Euler reducerat problemet till en funktion i en variabel. Vi har efter tidsstegningen hastigheten både som en funktion av tid och som en funktion av position. Kör också `total_consumption` med den nya ruttfilen.

(b) Lösning med ODE-lösare från SciPy: Kan SciPys ODE-lösare användas för att implementera en "route traveler"? Diskutera om det finns några problem med detta och om de skulle kunna lösas.

Tester och referens

I zip-arkivet `roadster.zip` finns en fil `test_roadster.py` som innehåller funktioner för att testa en del av koden från P1-P4. Det enklaste sättet att köra testerna är att använda sig av pytest. Testerna kan antingen köras från ett terminalfönster eller från en Python-prompt, tex i en utvecklingsmiljö som Thonny.

Tester i ett terminalfönster: När du vill köra testerna för en viss del, navigera till katalogen `roadster`. För att köra testerna tex för P1 eller P1a:

```
pytest -k part1
pytest -k part1a
```

och motsvarande för de andra delarna. Om du vill köra samtliga tester kan du köra `pytest` utan input.

Tester från Python-prompten: Se till att du står i katalogen `roadster`. Ladda först in modulen:

```
import pytest
```

För att köra testerna tex för del 1 eller del 1a:

```
pytest.main(["-k","part1"])
pytest.main(["-k","part1a"])
```

och motsvarande för de andra delarna. För att köra samtliga tester:

```
pytest.main()
```

Pytest fungerar så att den identifierar Python-filer som börjar med `test_` eller slutar med `_test` och kör alla funktioner däri som börjar med `test_` eller slutar med `_test`. Flaggan `-k` används för att endast köra de funktioner vars namn dessutom innehåller strängen som angetts. Om du har flaggan `-s` till pytest så skrivs `print`-sateser i din kod ut i terminalen (de visas inte annars), kan vara bra för debuggning.

Noggrannhetsordning: Låt $\tilde{u}_h$ vara en numerisk approximation av ett exakt värde u, där h är steglängd eller liknande i en numerisk metod. Den numeriska metoden har noggrannhetsordning p om det finns ett tal C oberoende av h så att

$$|\tilde{u}_h - u| \leq Ch^p, \tag{4}$$

åtminstone för tillräckligt små h.

Analysens fundamentalsats: Antag att funktionen f är kontinuerlig på intervallet $[a, b]$ och låt

$$F(x) = \int_a^x f(t)dt, \quad a \leq x \leq b \tag{5}$$

Då gäller att $F'(x) = f(x),\ x \in (a, b)$.